ENSURING CONSISTENCY AND EFFICIENCY OF THE INCREMENTAL UNIT NETWORK IN A DISTRIBUTED ARCHITECTURE

by
Mir Tahsin Imtiaz

A thesis

submitted in partial fulfillment

of the requirements for the degree of

Master of Science in Computer Science

Boise State University

May 2021

BOISE STATE UNIVERSITY GRADUATE COLLEGE

DEFENSE COMMITTEE AND FINAL READING APPROVALS

of the thesis submitted by

Mir Tahsin Imtiaz

Thesis Title: Ensuring Consistency and Efficiency of the Incremental Unit Network in a Distributed Architecture

Date of Final Oral Examination: 12th March 2021

The following individuals read and discussed the thesis submitted by student Mir Tahsin Imtiaz, and they evaluated the student's presentation and response to questions during the final oral examination. They found that the student passed the final oral examination.

Casey Kennington, Ph.D.	Chair, Supervisory Committee
Bogdan Dit, Ph.D.	Member, Supervisory Committee
Steve Cutchin, Ph.D.	Member, Supervisory Committee

The final reading approval of the thesis was granted by Casey Kennington, Ph.D., Chair of the Supervisory Committee. The thesis was approved by the Graduate College.

ACKNOWLEDGMENT

I must acknowledge my adviser, Dr. Casey Kennington, for his immense support and guidance, which included the graduate assistantship that supported my research and my pursuit of a Master's degree in Computer Science at Boise State University. I must also acknowledge that not only was he an adviser but also a friend throughout my entire degree. His constant advice, attention, and time is the reason I was able to come this far and complete this thesis. I would also like to thank the members of my advisory committee Dr. Bogdan Dit and Dr. Steve Cutchin for providing me with necessary feedback and advice that shaped this thesis.

I would also like to thank the other professors here at Boise State that I have interacted with. I must begin with Dr. Michael Ekstrand, who provided me with the proper knowledge in the area of Data Science that got me started here at Boise State. I also thank Dr. Gaby Dagher, Dr. James Buffenbarger, Dr. Amit Jain, Dr. Francesca Spezzano, and Dr. Edoardo Serra for helping me becoming a better Computer Science student, programmer, and researcher.

I would also like to thank the current and past members of the SLIM research group who welcomed me in and helped me grow. Every SLIM group member taught me something valuable during my time as a member of the group. I feel glad and privileged to meet and work with all these talented people. I also would like to thank the few people I met in Boise that I can call friends for inviting me in their lives and making me feel at home.

Finally I would like to dedicate this thesis to my family who supported me in my decision of taking a big leap and travelling half way across the world to study abroad.

ABSTRACT

An incremental system takes advantage of upcoming data as early as possible. In other words, an incremental system processes received data incrementally. Incremental systems can be useful over non-incremental systems to build spoken dialog systems when we are looking for faster and more human-like behavior. For example, human-to-human conversations are incremental, as a listener does not wait for a speaker to finish speaking to begin understanding. Inspired by the fact that Robot-Ready Spoken Dialog Systems must be incremental and need to work distributedly, and IU framework "breaks" in a distributed architecture, I attempted to use the IU network to fulfill the incremental requirements and be able to extend the IU framework to work flawlessly in a distributed environment. This work aims to answer the question whether we can make a distributed IU network efficiently and consistently. More specifically, I explored the optimal ways to establish a complex IU data store that can facilitate the conservation and accessibility of the total generated IU data network in a distributed environment avoiding the "breaking" of the IU network, and act as a backbone for a final and complete "Robot-Ready" incremental dialog system. We evaluated the HRI response differences happening along with IU store implementation differences in a live, interactive study with robots and found out that humans do notice small performance differences and subconsciously become judgmental of robots' anthropomorphism characteristics in relation to the robots' performance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ASR	Automated Speech Recognition
GRIN	Grounded-in Links
IU	Incremental Unit
Mask RCNN	Mask “Region Based Convolutional Neural Network”
NLU	Natural Language Understanding
POS	Part of Speech
PSI	Platform for Situated Intelligence
SDS	Spoken Dialog Systems
SLL	Same-Level Links
WAC	Words-As-Classifiers Model of Lexical Semantics

CHAPTER 1: INTRODUCTION

As robots become more commonplace, there is an increased expectation that humans will interact with them. The most natural means of communication between people and robots is not with keyboards or other common interaction devices (e.g. touch screens), but with spoken dialog since robots are intended to be used by people with little or no computing experience [1]. In a situation where humans and robots are partners, collaborative dialog can evade a lot of resource-consuming maneuvers just by having a natural communication as humans do with others [2]. Therefore, by extension, I posit that dialog will be the eventual common way for humans to communicate with robots.

This thesis is concerned with the infrastructure for making spoken dialog systems (SDS) more "ready" to work with different robot platforms. Specifically, we explored the ways to make a dialog system achieve all the desired capabilities while keeping the infrastructure consistent and efficient as well as minimizing the workload of rebuilding a new framework to facilitate all of the requirements. We brought in two existing frameworks each having a subset of the desired capabilities for a robot-ready spoken dialog system in order to make them work together building one common robot-ready spoken dialog system that works consistently by holding all the information without breaking the overall network. Namely, we used PSI [3] which is a framework from

Microsoft for building multimodal, integrative AI systems and ReTiCo [4]) which is a framework for construction of incremental spoken dialog systems. We explored different ways of building and holding a common robot-ready incremental network using the two frameworks and evaluated their performance based on stress tests and live human-robot interaction experiments.

1.0.1 The "Robot-Ready" Dialog System

Following [5], a standard dialog system for use in an embodied agent (e.g. a robot) should fulfill the following requirements:

- **modular:** the system is composed of multiple modules, and new modules can be integrated with the system
- **multimodal:** the system can take in and integrate inputs from multiple sensors
- **distributive:** modules are able to communicate in distributed environments flawlessly (i.e. communication should be reliable)
- **incremental:** modules in the system process received inputs quickly and instantly
- **temporally aligned:** multiple sensor inputs must be aligned with each other (e.g. aligned with respect to time)

When it comes to facilitating the infrastructure of systems that fulfill the above requirements, there have been several implementations that are multimodal including PSI [3] and Pythia [6], each written in different programming languages for different environments or for particular purposes that helps with the development and study of complex, multimodal AI systems. What is meant by incremental processing is that speech input is processed word-by-word. For research purposes in the area of dialog systems, which is inherently incremental [7], multiple incremental frameworks have been developed (e.g. InproTK [8], ReTiCo [4]) that support incremental processing and can be used to build complex speech-based human computer interfaces. Although the frameworks developed so far are useful to the research community, no single framework exists that fulfills all five requirements mentioned above and ensures the consistency of the incremental network both within and outside of that particular framework. However, two different implementations put together, PSI and ReTiCo, could fulfill all five requirements given that we solve the preservation problem of the data network generated by a given IU network for a distributed architecture.

The problem we seek to address in this thesis is whether we can bring multiple incremental processes together merging their incremental properties so that they communicate with each other efficiently and consistently, and preserve the incrementality in the process. In the following section, I give additional necessary background for incremental processing and how it might be negatively affected by a distributed environment. Then I give background about PSI and ReTiCo, and explain how they could potentially, taken together, fulfill all five requirements.

1.1 Background

1.1.1 Incremental Processing

An incremental system takes advantage of upcoming data as early as possible. In other words, an incremental system processes the received data incrementally [9]. Instead of waiting for all information to come in, the modules in an incremental system start processing as soon as they start getting the minimal amount of input [10] from the previous modules, taking advantage of being able to output smaller chunks and updating it as more information is revealed as input later on. The minimal amount of input or data that an incremental module receives is called Incremental Units (IU) [11]. An IU framework [7] is a structural and conceptual approach of implementing incremental systems based on IUs.

A typical IU network consists of multiple IU modules each having a left and a right buffer. Modules receive data as IUs from the previous module using its left buffer, perform computation on the received data, and pass the newly processed IUs using the right buffer [12]. Each of the IUs in a particular module is connected using Same-Level Links (SLL), whereas IUs from different modules are connected using Grounded-in Links (GRIN) (more explanation is given below).

Since incremental networks work by processing data as early as possible with the amount of data at hand, they need some kind of mechanism to update IUs based on updated information from time to time. ADD, REVOKE, and COMMIT are the three main types of operations we can perform on IUs [8]. ADD is the operation of adding a new IU when new information is available. As new information keeps being revealed, modules may determine that information of some particular IUs previously generated may not be relevant or useful anymore. In this kind of case, an incremental module REVOKES a previous IU as well as let other modules know of its action because some modules may have already processed those IUs. Finally, when it is determined that certain IUs added to the network have no chance of being changed or updated and is the final result at that time, modules mark that IU as a COMMIT.

A classic example of an incremental module is an incremental speech recognizer

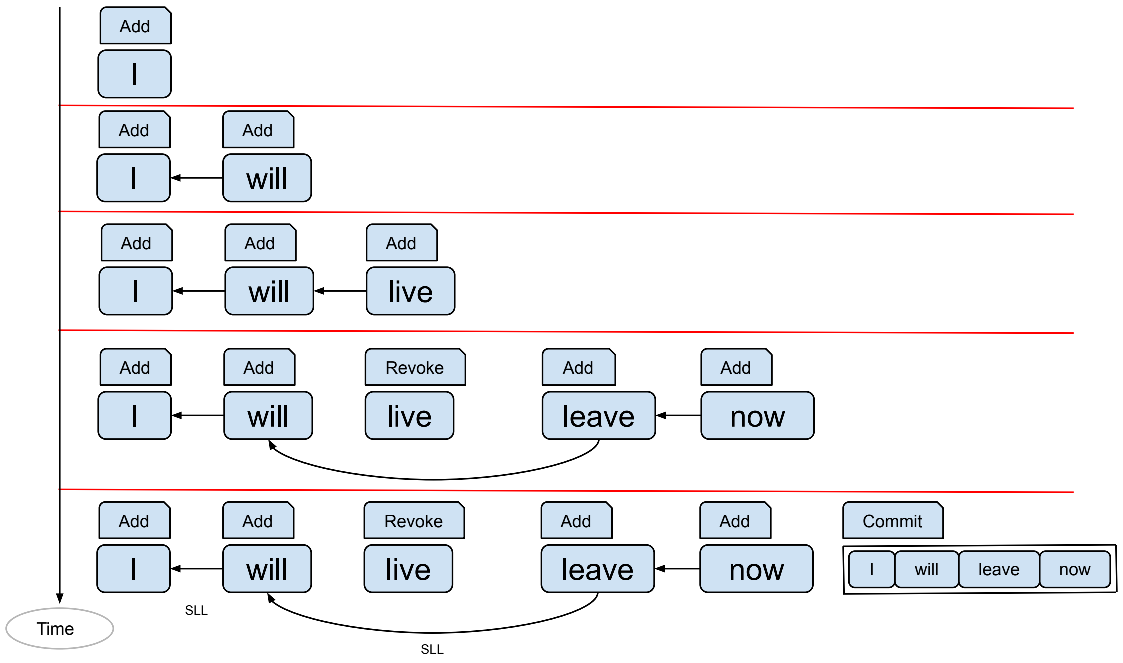

Figure 1.1: Example of SLL and Add, Revoke, and Commit operation for an incremental speech recognizer.

module which processes output word-by-word rather than waiting for silence or the user to finish the entire utterance. Figure 1.1 shows how IUs are connected together in an incremental speech recognizer module using Same-Level Links and how IUs are added, revoked, and added further as more information becomes available from the previous module (a microphone module listening to user utterances). The speech recognizer initially predicts the partial results as “I will live”. However, after getting more information, it realizes that the result is actually “I will leave now”, revokes the IU “live”, adds the IUs “leave” and “now”, and finally commits the final result as “I will leave now”. Figure 1.2 shows how IUs of an ASR (Automated Speech Recognition) module and POS (Part of Speech) module are connected using Grounded-in Links.

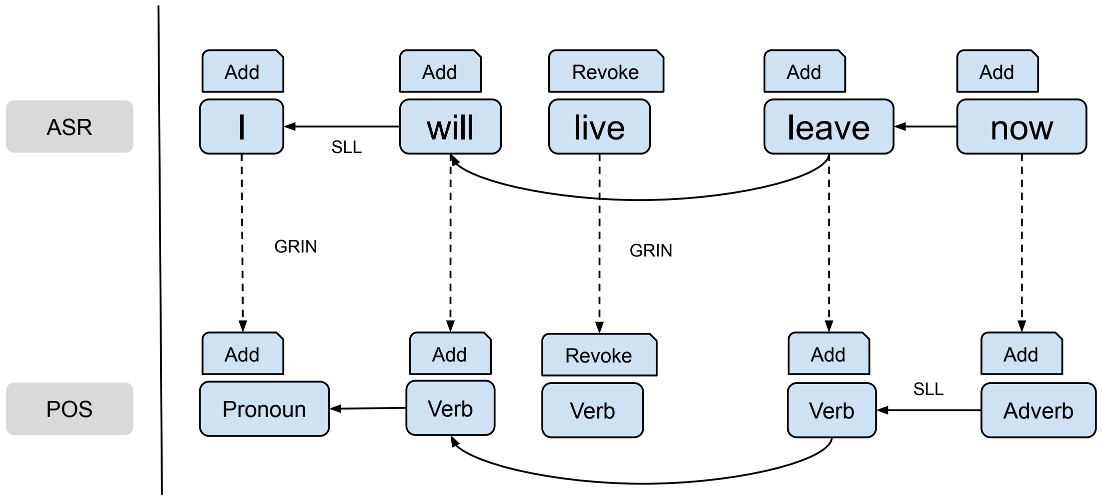

Figure 1.2: IUs from two modules (speech recognizer and parts of speech tagger) connected by GRIN.

Incremental systems can be useful over non-incremental systems when we are looking for faster and more human-like behavior. For example, human-to-human conversations and language processing is incremental, as a listener does not wait for a speaker to finish speaking to begin understanding [13], and incremental systems perform relatively faster than non-incremental systems since components or modules work simultaneously instead of waiting for a module to complete its entire processing [14]. Since an incremental system works with a minimum amount of input, it is expected that an incremental dialog system will be able to capture behaviors like concurrent feedback, fast turn-taking, and collaborative utterance construction, which is not possible for a non-incremental system [15].

1.2 IU Network in Distributed Environment

If we look at Figure 1.1 and Figure 1.2, and inspect how information is flowed and shared through the modules of an IU network, we can see that all the modules in an IU network create a complex network of IUs together. As mentioned above, the IUs in a particular module are connected using Same-Level Links (SLL) whereas surrounding modules maintain connection with each other's IUs using Grounded-in Links (GRIN) resulting in the complex network of IUs. Any module can query other modules starting from a particular IU object to find out its history just by traversing this network. This network of IUs can be realized more from Figure 1.3. If these modules are situated in different systems, we use mechanisms like inter-process communication to establish the network and use standard message format like JSON or standard binary format like Thrift [16] to send the appropriate data. However, for the IU network, this is not enough because when there are such connections between two modules situated in different systems, the destination module loses the capability of traversing back to certain points in the IU network that are situated in the other system since the object properties connecting it to the remaining network get lost. In other words, an overall incremental system consisting of modules situated in different processes faces breaking the overall IU network although a minimum level of connection is achieved, and certain jobs are completed by the system.

1.3 Inter-process Communication

One of the issues we need to address is how do we make two incremental processes talk to each other? In other words, how do these two systems exchange necessary information (relevant IUs and how they are connected to the other IUs) so that they work flawlessly? This is where interoperability comes in, which is the ability of two implementations of systems or components from different processes to co-exist and work together by only relying on each other when necessary [17]. If we consider our whole project as one full system, "Interoperability" is the term that is used to refer

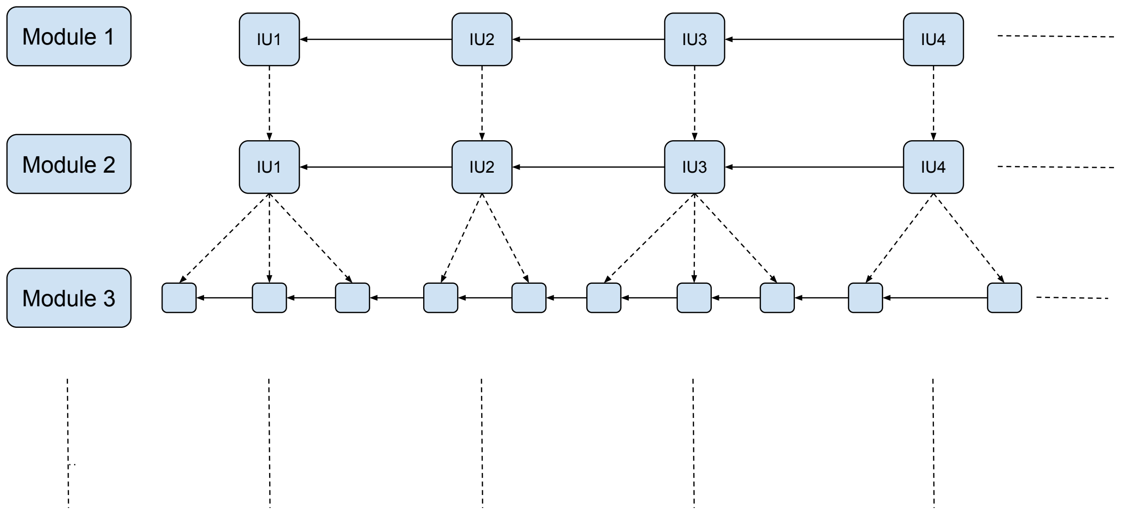

Figure 1.3: A complex network of IUs generated by multiple modules.

to the idea of using two processes in two programming languages together to the necessary extent [18]. It is important that we find a common and conventional way that works efficiently given the systems in question. Programmers often enable the use of existing libraries written in another language using interoperability techniques since it is not always feasible rewriting whole libraries in the new language due to its large and complicated implementation, and interfacing with the existing implementation is the more realistic approach [19]. For example, dotnet manages this with C++ by supporting direct interoperability with only a subset of C++ (Managed C++ or C++/CLI) [20]. Python also provides foreign function interface support for running extension modules which are written in lower level languages such as C, although the support is limited for modern implementations of dynamic languages [19]. We can also use message-based inter-process communication instead of composing language implementations at their implementation level. Examples of this kind of implementation supporting message passing between systems written in different languages are

Protocol Buffers [9] from Google and Thrift [16] from Facebook. While using these types of implementations, engineers and programmers work with a language-agnostic interface definition language. This kind of interface marshals the data into a common representation that can be interpreted on both sides.

In order to maintain the entire IU network while two processes are running and communicating using an inter-process communication technique, we need to store and update the network from the perspective of both the processes in some kind of logical shared database. The performance will depend on how we send data between the two processes and how we implement this shared database. While sharing data with each other, we can keep sending reference to the entire data-structure or only send the new chunks of data. In addition to that, we can implement the database in one of the processes where the other process can query for relevant data (e.g. PSI holds the database and ReTiCo queries to PSI), or we can implement a shared database where both processes will have access to (more details and explanation about the implementation strategies are given in the experiment subsection). In the next three sections, we will briefly introduce our two target frameworks, PSI and ReTiCo, and compare them in terms of their features.

1.4 Platform for Situated Intelligence (PSI)

PSI [3] is a framework from Microsoft written in C# that opens the door to easy development and study of multimodal and integrative AI systems. In spite of not being written in one of the most popular programming languages in Machine Learning [21], PSI works by providing a parallel programming model centered around data streams, enables easy development and connection of components while keeping the performance properties of a natively written system, and encapsulates various AI technologies allowing quick composition of complex AI applications.

One of the main reasons to bring PSI into our research is the properties and features it affords. PSI already fulfills some of the requirements for robot-ready SDS as mentioned above. Moreover, the framework provided by PSI is time-aware and has the capability of meaningful stream fusion which is one of the core requirements for an incremental system depending on the network topology [9]. This feature has not been introduced before in any other frameworks as efficiently as PSI. In addition to this, PSI brings in tools and APIs enabling multimodal data visualization and analysis in real time.

1.5 ReTiCo

ReTiCo [4] is an incremental framework written in Python and enables the construction of incremental spoken dialog systems providing a wide range of incremental modules (e.g. Rasa NLU, PyOpenDial Dialog Manager, etc.). The framework is user-friendly and allows construction of a network with a few lines of code initializing modules and connecting them according to their left and right buffers.

While ReTiCo is a classic example of an easy-to-use standard incremental framework based on spoken dialog systems, it is missing some of the key features required for research in incremental, modular, and multimodal systems. Namely, it is missing the appropriate mechanisms that can work with concurrent data streams coming from different modules and standard data storage facilities for data analysis.

1.6 PSI and ReTiCo Features Comparison

As mentioned already, standard dialog systems should fulfill certain requirements although it is not necessary to achieve all of them in order to achieve a fully incremental dialog system. Table 1.1 shows a comparison between PSI and ReTiCo in terms of these requirements.

Table 1.1: Features comparison of PSI and ReTiCo.

Requirement	PSI	ReTiCo
Modular	Yes	Yes
Multimodal	Yes	Yes
Distributive	Only data passing using third party	Only data passing using third party
Incremental	Logically	Logically and Structurally
Temporally Aligned	Meaningful stream fusion w.r.t time	No

As shown in Table 1.1, neither PSI nor ReTiCo fulfills all the requirements completely. While PSI holds the temporal alignment capability, it does not hold incrementality by following a standard structure. ReTiCo on the other hand is structurally incremental which means IUs are passed on between modules following a standard. However, it cannot achieve temporal alignment. In addition to that, both PSI and ReTiCo only support distributive features by allowing data passing in JSON only. There is no structure or mechanism to facilitate structural connection between modules or hold overall IU network output consistent. As a result, in the event of an actual distributed setting, both lose the ability to build a genuine shared IU network, and they can only be used for certain simple jobs (e.g. one building an incremental network, the other storing the final result as a logger).

1.7 Thesis Statement

How can we use the IU network to fulfill the incremental requirements and be able to extend the IU framework to work flawlessly in a distributed environment? My work aims to combine multiple complex incremental processes so that they work together to maintain a common IU network across the two processes, taking advantage of each other's strengths (properties only one of the processes have that both can utilize) to do complex tasks that they could not do alone. More specifically, I explored ways to answer the question whether we can make two multimodal frameworks come together while making the best use of both, adding incremental properties to one, then merging their incremental capabilities, avoiding breaking the IU network, ensuring accessibility of the full network to both frameworks, and maintaining efficiency and consistency of the incremental network in the distributed environment while building a complete "Robot-Ready" Dialog System. Moreover, I explored the optimal ways to establish a complex IU data store that can facilitate the conservation and accessibility of the total generated IU network in a distributed environment avoiding the "breaking" of the IU network, and act as a backbone for the final and complete "Robot-Ready" incremental dialog system. When it comes to building large and scalable software systems, a better way for managing software complexity is using a collection of components ensuring reusability, modularity, and fault isolation [22]. In software development, it is common for programmers to use the most suitable language for a particular job, combining different sets of languages, and reusing existing source code [23]. This allows them to pick the best system for particular tasks while accomplishing an overall complex job.

This thesis seeks to apply the similar approach for standard incremental systems to achieve a complex task combining tool sets from different environments. Previous research already attempted to use multiple systems together that share common interoperability techniques where one system contributes into building the major portion of the network, and the other system does rather a small particular portion of the overall task [5]. The IU network already has the theoretical potential to work in a distributed environment without breaking from previous research in this area [7]. I hypothesize that the proposed approaches of constructing a distributed IU network along with building a custom IU store that holds the overall data facilitating the conservation of IU data generated by the network will be consistent and efficient, and overall, the proposed system should motivate further research consisting of distributed IU networks.

We designed an experiment where we brought in two incremental frameworks with different sets of capabilities, together performing a complex task (more details about the overall approach are in the following chapters) of a robot-ready spoken dialog system. We used systematic approaches to evaluate the final implementation and used the Cozmo[1] robot to interact with human partners to understand the percept of humans of our different approaches. The humans asked questions and gave commands to the robot, expecting appropriate answers or actions as feedback. The robot used our two processes in the background together to get done different parts of the overall task. This, along with our systematic evaluation, assessed the efficiency and consistency of the implementation.

The problem we addressed in this thesis is important because it gives us insight on building consistent incremental networks that can facilitate a stable increment of data inside incremental modules that are not only isolated in one place, but can be dis-

[1]https://www.digitaldreamlabs.com/pages/cozmo

tributed in different machines without breaking the network, which is a common issue in multimodal incremental systems. Hopefully, this study will enable and motivate us to build more complex and efficient incremental systems that will eventually lead to more advanced research in this area. Merging the advantages of multiple frameworks of a particular area and facilitating easier access to complex systems can accelerate research and development as well as make more people interested in working in those areas.

CHAPTER 2: RELATED WORK

Prior research in the area of AI, robotics, and dialog systems demonstrated the use and advantages of working with or building on existing systems and connecting them together. MultiBot [24] was built by using and leveraging already-existing components from ScoutBot [25] by extending the mode of interaction to multi-participant dialog. In the development of smart office space facilitating collaborative learning, Wang et al. [26] used multiple software systems to integrate their capabilities where the Bazaar toolkit [27] was used for the foundation for the dialog-based support offered within the space, and PSI was used for coordination of data streams. In general, Bazaar was mainly used as an extension module, and in order to ensure proper message passing between the two tools, they used an internally developed multimodal message format which consisted of any combination of location, speech text, body position, facial expression, and any detected emotion.

Kennington et al. [12] extended the incremental processing toolkit InproTK [8] to InproTKS in order to enable it to receive multimodal sensor data, and achieve situated and real-time dialog. Being an incremental system, InproTKS was another potential candidate for this thesis that focuses on building a complete and ready dialog system. However, the two chosen frameworks (PSI and ReTiCo) bring together all the mentioned requirements. Namely, PSI brings in perfect temporal alignment of data, and

ReTiCo, being written in Python and already being incremental, represents multiple existing tools related to dialog systems. Kousidis et al. [28] created a multimodal In-Car dialog system by using the OpenDS [29] toolkit as a driving simulator and using InproTKS [12] to build the dialog system for their experiment. Although their dialog system was incremental, they used the different message passing techniques available to them (Robotics Service Bus (RSB) message passing architecture[1] [30], and InstantIO/InstantReality[2]) only for logging results in XML file format for further analysis. Carlmeyer et al. [31] combined InproTK [8] with PaMini [32] in order to allow closed feedback loops in HRI so that their interactive system can adapt to the user. In their experiment, they used Robotics Service Bus (RSB) as well. In order to structurally send and receive messages between InproTK and PaMini, they modified InproTK listener and informer so that appropriate dialog acts can be sent and incoming verbalizations can be split and processed as small phrases. Moreover, since PaMini only reacted to inputs with "COMMIT" state, they created a new input source for PaMini that reacts to dialog acts from InproTK. In other words, they attempted to solve the issue of connecting one external module (dialog manager PaMini) to the incremental framework InproTK. Kennington et al. [5] worked towards a robot-ready spoken dialog system by integrating multiple toolkits in ReTiCo. They only used PSI as a logging tool in order to take advantage of its data storage utilities and have not utilized its temporal alignment capabilities. Although all this prior work deals with building multi-framework systems (incremental or non-incremental), they have not built their systems from the IU network perspective. In other words, they did not focus on resolving issues with the consistency of the IU network in the distributed

[1]https://code.cor-lab.de/projects/rsb

[2]https://www.instantreality.org/

environment–the goal of this thesis.

CHAPTER 3: METHOD

In order to show that a complex incremental network can be constructed and used efficiently using two incremental frameworks in a distributed environment preserving and maintaining the IU network, we ran a similar experiment as mentioned in [5] where a robot receives data from a microphone and camera and the overall processing of the total pipeline is done using two of our frameworks together.

We used PSI and ReTiCo, one of which is an open, extensible framework written in C# enabling the development and study of integrative AI systems, and the other is a Python framework which is based on [7] and enables the construction of incremental spoken dialog systems. In our experiment, we made them work together building a common incremental network as shown in Figure 3.1. The details of the construction are in the following sections.

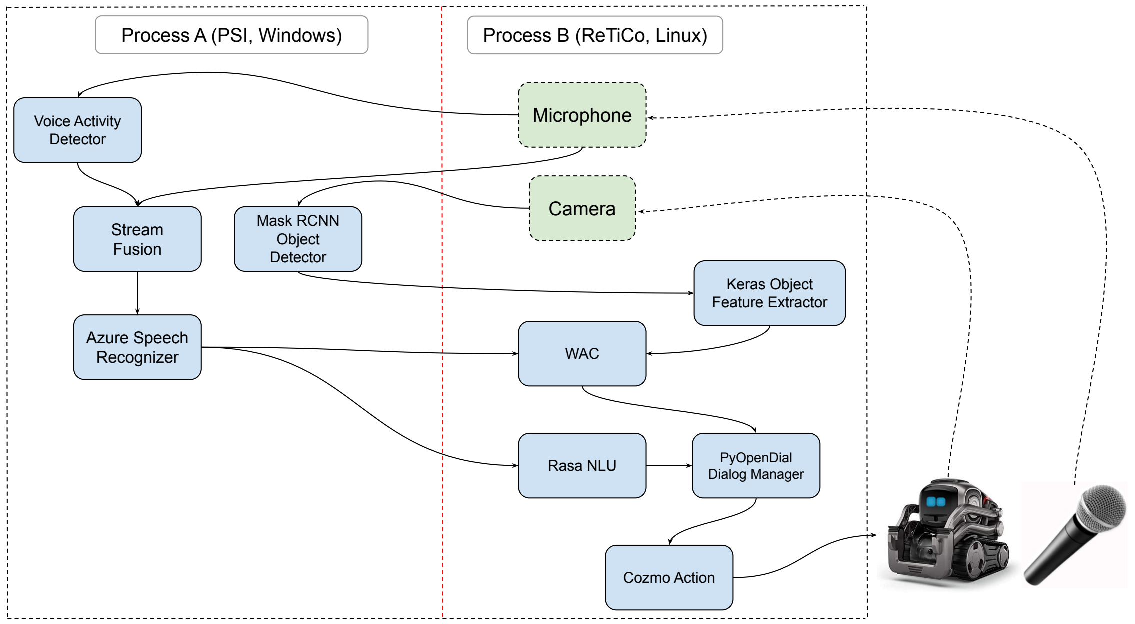

Figure 3.1: Overview of the multi-framework, multimodal, incremental, distributed network; the two processes communicate with each other using a message passing bus; the goal of this thesis was to maintain a shared data structure required for incremental processing that both of the processes can access.

3.1 Preparing Platform for Situated Intelligence (PSI)

Although PSI already works incrementally in that it can handle continuous intput, it lacks one of the most basic requirements of communication in an incremental network: a standard way of breaking down or dividing up larger units of data into smaller ones. In other words, PSI needs to be made to work within the IU framework (something that ReTiCo does natively). To resolve this issue, we have implemented a deltafier that takes in the smallest possible data and packages the data as an IU. Each IU is composed of an EditType which explains whether the data is being Added, Revoked or Committed, TimeStamp which is a variable of type double holding the Epoch time[1] during the creation of the IU (although the ToString method will convert that to a Date and Time as shown in Figure 3.2), a reference to the relevant previous IU (SLL) in the module that has not been revoked, a reference to the Grounded-in IU, and the payload of that IU. A payload for a speech recognizer is a text (e.g. a word) holding the recognition result. The entire data generated in a module is stored and held as a list of IUs and each module defines its own IU and payload. Figure 3.2 shows this structure for one IU of a speech recognizer.

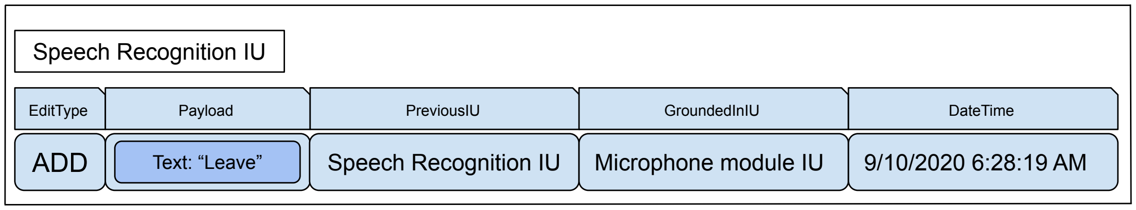

Figure 3.2: A general structure of one speech recognition IU.

3.2 PSI Modules in the Network

As shown in Figure 3.1, the total network has been split between the two frameworks, each doing a fair share of the processing. In the network, PSI is responsible for voice activity detection, meaningful stream fusion of incoming foreign and native data, object detection from image data, and speech recognition from audio data.

3.2.1 Joining or Fusing Data

Being able to join or fuse streams appropriately (temporal alignment of data with respect to time) is a unique feature of PSI that ReTiCo does not possess. In our experiment, PSI was in charge of fusing the incoming microphone stream received from

[1]https://en.wikipedia.org/wiki/Unix_time

ReTiCo with the voice activity detection stream generated in PSI. This demonstrates that meaningful stream fusion can be done in the distributed environment.

3.2.2 Speech Recognition

The PSI was responsible to implement the speech recognition module. We used Azure Cognitive Speech services to implement a speech recognition module that worked along with the deltafier mentioned above to perform incremental speech recognition.

3.2.3 Object Detection

PSI object detection module uses the Google MaskRCNN [33] for proper object detection on camera streams received from ReTiCo in order to pass it again back to ReTiCo for feature extraction.

3.3 ReTiCo Modules in the Network

ReTiCo has existing implementations of modules to perform common tasks related to dialog systems. Therefore, ReTiCo is used in the network to support those segments. Specifically, ReTiCo is responsible for reading sensor inputs (microphone and camera), natural language understanding, dialog management, and sending the final signal to the robot to demonstrate proper action.

3.3.1 Rasa NLU (Natural Language Understanding) Module

The Rasa NLU module in ReTiCo receives speech reccognition IU from PSI and generates appropriate dialog act IUs containing the appropriate act and concepts. This module is responsible for primarily controlling all the actions of the robot that are related to language inputs only. The output from Rasa NLU is sent to PyOpenDial [34] module for proper dialog management.

3.3.2 Keras Object Feature Extractor and WAC

The Keras Object Feature Extraction module takes in IUs from Google MaskRCNN in PSI to produce a vector that is fed into to words-as-classifiers (WAC) [35] model. In addition to the IUs from Keras Feature Extractor, WAC also receives speech recognition results from PSI. The WAC model or word-as-classifiers model is a grounded model of lexical semantics that can link words to the physical world. In other words, WAC is a way to map the visual world to the user utterance. Therefore, it is useful when the robot has to interact with the physical world in response to a command or a question. For example, when a participant says “Is the block red?”, the vectors generated by Mask RCNN is fed into WAC to decide whether “red” is the best answer for the particular object, and the robot action is generated based on that information. Figure 3.3 shows the idea of word grounding into physical features and how that decides the robot action. The output from WAC is also sent to the PyOpenDial module for dialog management.

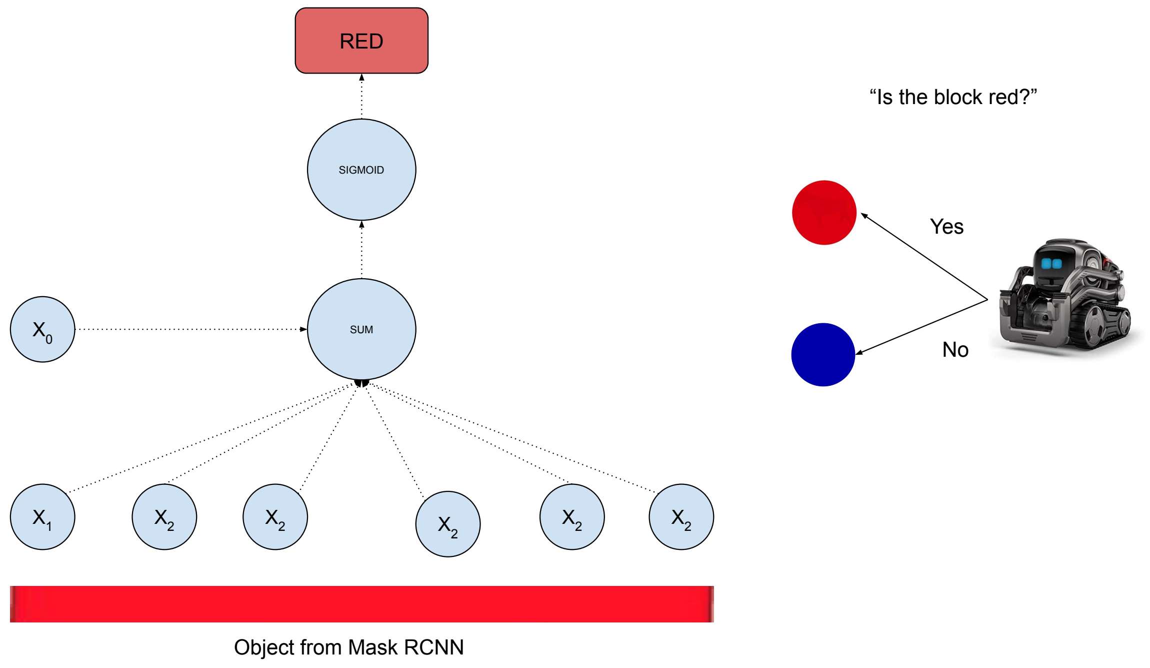

Figure 3.3: Words grounding into physical modalities affecting robot action.

3.3.3 The PyOpenDial Module

PyOpenDial [34] is a dialog manager that controls the final behavior of the Cozmo robot in response to a query. It utilizes speech recogniztion IUs from PSI ASR, dialog act IUs from ReTiCo RASA NLU, and grounded frame IUs from ReTiCo WAC either directly or by grounding into the IU network when necessary. In others words, among all the other modules, this module most leverages the capability of traversing the distributed network using the IU store to access relevant IUs to make its decisions. This module signals the Cozmo action module to make the robot perform certain appropriate actions.

3.3.4 Interoperability Between the Two Frameworks

One of the core requirements for our overall framework to work is the interoperability of the two target systems, or how they communicate with each other (as per the distributive robot-ready requirement). In our experiment, we mainly attempted two options for achieving interoperability. Namely, we evaluated different foreign function interface options that are available for C# and Python, and message queue implementations.

In terms of foreign function interfaces, it would have been helpful and easier in terms of implementation if these libraries supported all the complex functionalities required by our two incremental frameworks. Unfortunately both for .NET platform and Python, these interfaces only support a limited subset of certain low-level languages (e.g. C++). There are also open source projects that help with the integration of Python and the .NET platform such as Python.NET[2] and IronPython[3]. While Python.NET provides seamless integration with the .NET Common Language Runtime (CLR), and IronPython is a good candidate being an implementation of the Python programming language targeting the .NET Framework, none of these projects can access the Generic Extension methods[4] that PSI uses to actually create and complete a pipeline network since references to these functions are resolved during compilation time and are not possible to get access to during runtime.

The next logical option to achieve interoperability is using message queue since both PSI and ReTiCo support message passing using ZeroMQ, which is a universal

[2]https://github.com/pythonnet/pythonnet
[3]https://ironpython.net/
[4]https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/extension-methods

message passing library. It builds and maintains sockets that carry atomic messages across various transports. Both frameworks use JSON format to send data using ZeroMQ.

3.3.5 Module-to-Module Interoperability in the IU Network

From previous research, we have witnessed attempts of solving module-to-module connectivity. However, they mostly apply to connections between two specific modules [26; 28] and do not specify or explain any general structure. As mentioned above, our experiment consists of multiple module-to-module communication in the distributed environment. Therefore, for our experiment to work it is important that both our target framework does similar IU packaging while sending IUs from one module to another and follows a certain structure during interoperability to achieve flawless connection. By default, ReTiCo has standard IU packaging for its modules which contains payload information, previous IU linked in the network, the Grounded-in IU information, and age. PSI, on the other hand, did not initially have similar IU packaging. As mentioned above, we have already implemented that functionality. Currently PSI holds IUs as a list where each element in the list is an IU-holding payload, reference to previous and Grounded-in IU, and timestamp, which is similar to ReTiCo.

In order to convey this information, every time we interop from one module to the other that is situated in another framework or process, we need to make sure this structure does not get lost or broken and all the Same-Level Links (SLL) are preserved. To guarantee that, we need to ensure that we treat the network elements as objects [7] and use a middleware module in the destination framework that will do the job of appropriate data-to-object conversion and reestablish all the same level connections in the receiving framework before passing the data along to the next module. I implemented this middleware in two ways considering the effects in terms of consistency and efficiency.

The first approach required sending a reference to the entire SLL network every time new IUs are generated. This ensures maximum consistency but sacrifices efficiency. In this approach, efficiency is made better by only parsing the new IUs and ignoring the old ones in the destination middleware. The second approach focuses on efficiency by only sending the newly generated IUs. In this approach, the middleware achieves consistency by rebuilding the entire SLL network at the destination in order to reestablish the object connections and keep track of the history.

3.3.6 Conserving and Maintaining the Entire Distributed IU Network

The structures mentioned in section 3.3.4 ensure that module-to-module communication is established by properly transferring IUs and preserving their Same-Level Links (SLL). However, in terms of Grounded-in Links (GRIN), interoping IUs from one system to another results in losing reference to the part of the network that is situated in the other process or machine since most of the network properties get lost resulting in breaking the overall network. It is infeasible to try to traverse the network when different parts of the network are in different environments. In order to solve this dilemma, we attempted to implement an IU data store that holds onto the entire IU data generated by the modules in the processes. Regardless of the type and implementation strategy, all of our IU store implementations (implementation details follow) have the following main functions:

- **InsertIU:** This function inserts an IU to the store.

- **RetrieveIU:** Given the ID or key of the IU, this function returns that particular IU.
- **GetPreviousIU:** Given the ID or key of an IU, this function returns the previous IU of that particular IU (e.g. giving the ID of the IU that holds the word "now" will return the IU that holds the word "leave" in Figure 1.2)
- **GetGroundedInIU:** This function returns the grounded IU of a particular IU given the ID of that particular IU (e.g. giving the ID of the IU that holds the payload "Adverb" will return the IU that holds the word "now" in Figure 1.2)

This way the modules in the different processes can use these functions to retrieve any data necessary for running the pipeline and need not to think about the complex implementation. These functions create a IU-storage-request (RetrieveIU function requests or queries for a "Retrieve" request) to the under-the-hood database to get the relevant data. In other words, the different types of IU store I implemented facilitates the logical traversal of the incremental network and thus the IU network gets conserved and avoids "breaking". I implemented two main ways to facilitate this logical IU data store for robot-ready dialog system pipeline as follows.

- **Storing IUs in PSI:** In our first approach, we chose one framework (PSI) that is responsible for holding the entire IU network as a database. Both the chosen framework (PSI) and the other framework (ReTiCo) send their IU network information to the database. PSI can store its IUs directly to the store whereas ReTiCo sends its IUs to PSI using message-passing IU store request. In this way, PSI holds and views the entire IU network directly whereas ReTiCo queries PSI to retrieve certain information from the network. (We treat our two systems as client (ReTiCo) and servant (PSI), and follow a Remote-write protocol[5]. The servant framework (PSI) holds a database storing the network from both the frameworks. Since there is only one server in our case, this approach enables us to achieve sequential consistency[6]. While the servant framework has direct access to the storage, the other client framework queries the server framework for relevant information.) We attempted two types of native implementation of the PSI storage: one in C# since PSI is written in C# and another in C++ to compare the efficiency of a relatively lower-level implementation with the native C# implementation and the second approach.

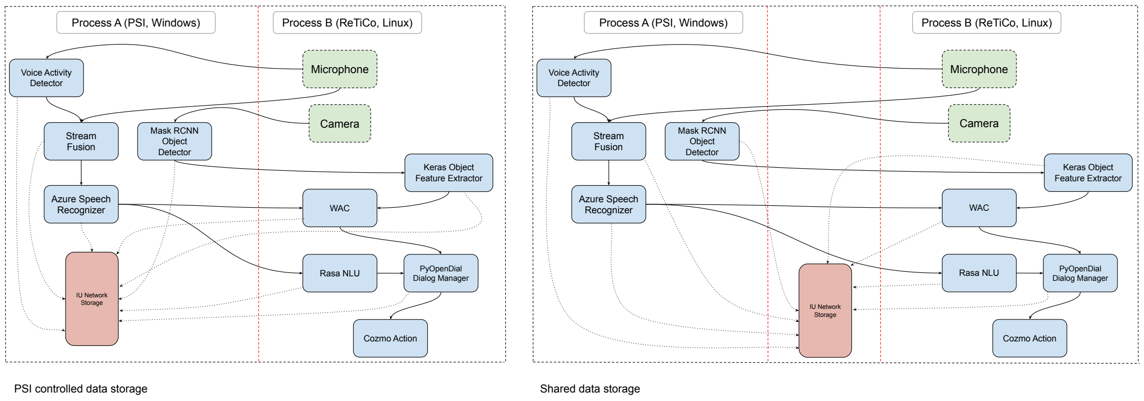

Figure 3.4: Distributed IU Network Storage Topologies.

- **Storing IUs in Redis:** In the second approach, we attempted a shared data storage to which both of our frameworks have access and can store their respective generated IUs. One of the fundamental issues in any distributed system is achieving sequential consistency with concurrent operations. In any given IU network, no IU is generated before its parent IUs and modules have obvious

[5]`https://en.wikipedia.org/wiki/Consistency_model#Remote-write_protocols`
[6]`https://en.wikipedia.org/wiki/Sequential_consistency`

latency between them. Therefore, an implementation of a shared data storage for a distributed IU network achieves sequential consistency, since we are only dealing with one logical data server. For this approach, we used the popular data-structure-project Redis [36]. These kinds of projects are known to be relatively fast, efficient, and support shared memory [37]. Figure 3.4 shows the comparison between the two approaches.

3.4 Final Overall Network Structure

Figure 3.1 shows the final network I have used in this thesis. We used two machines to setup our two frameworks. PSI was situated in a Windows machine (a Windows 10 laptop) whereas ReTiCo was situated in a Linux machine (a Ubuntu desktop). The native IU storage was within the Windows machine (within PSI whether it is C# or C++), and the Redis instance was hosted in the same Linux machine as ReTiCo. Both the machines were connected to the same private network. The Windows machine was connected using a Wi-Fi connection whereas the Linux desktop was wired connected to the network.

3.5 Evaluation Criteria

For the evaluation of our system with all the key combinations mentioned above, we mainly focused on consistency and efficiency of different parts of the system and the system as a whole. We divided our evaluation into two parts, one of which systematically evaluated consistency and efficiency of the network, and the other experiment used a live evaluation of the system with humans.

The first experiment focuses on two metrics: the first solely focuses on the consistency of the system. We tested whether the connections were consistent throughout the entire network. This consisted of checking whether each module-to-module communication is consistent (e.g. ASR output from PSI generating proper IUs in ReTiCo NLU without losing any data), the final output of the entire construction is correct, and the entire network is stored correctly. In order to do this, we generated unit tests for each module-to-module communication when the modules were situated in different systems. Unit tests were also generated to check the output of the overall system. Checking the final output using unit tests also ensures that the entire network is preserved and is consistent since some of the modules look for history in the IU network to perform correctly (e.g. PyOpenDial dialog management module asks for the speech recognition word associated with the Rasa NLU IU).

The second metric was based on determining efficiency. Although the entire network is consistent, we needed to check if the entire process is efficient and actually feasible for bigger networks. We evaluated our module-to-module and overall connection of the network by a similar approach to [38] and evaluated our implementation based on latency and IU delivery success rate.

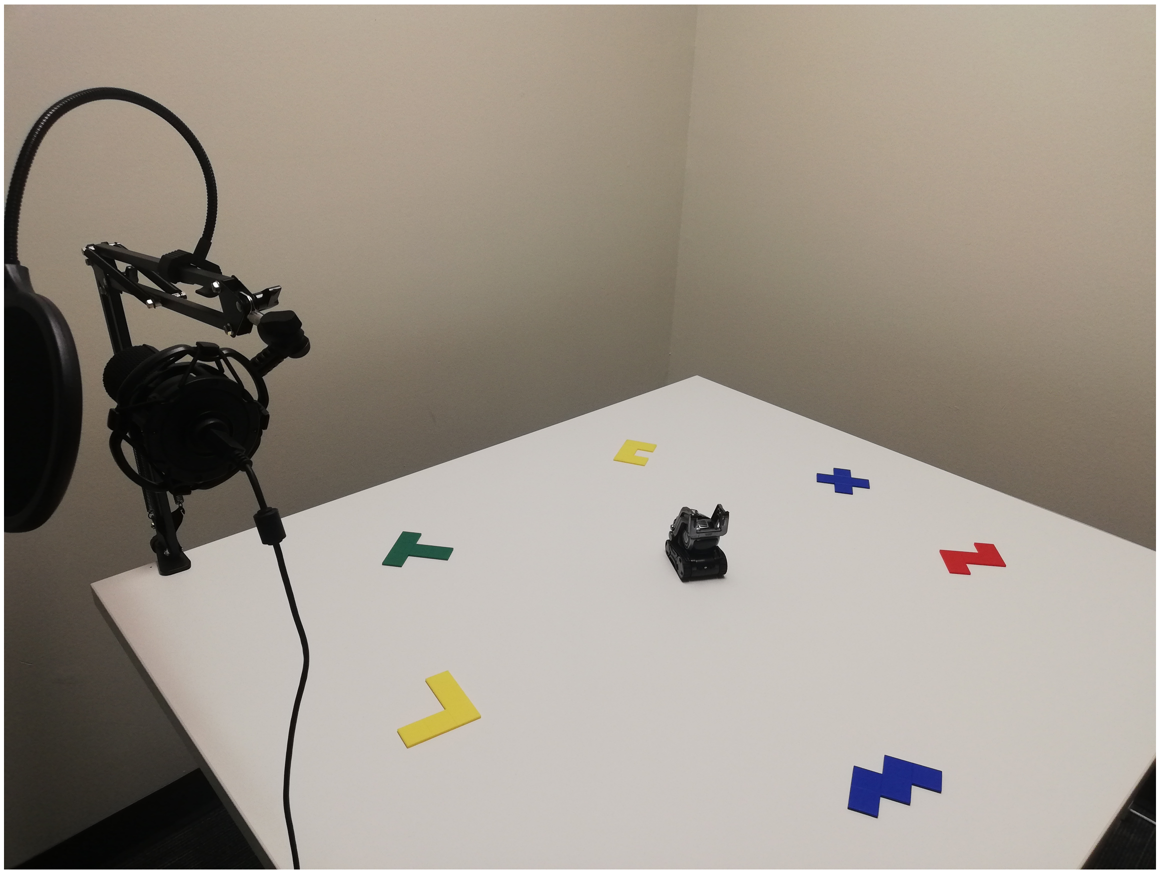

Figure 3.5: Cozmo robot in its task setting.

The second experiment is based on human interaction. Although the different metrics mentioned above show that my implementation effectively works as expected at an implementation level, it is important to evaluate how the full system mentioned above affects interaction with humans when we deploy the distributed robot-ready standard dialog system. In order to examine that, we designed a similar task setting as [5] using the Cozmo robot. Figure 3.5 shows the task setting of the Cozmo robot for our experiment design. Cozmo is surrounded by colored objects, can see its surroundings, listens to the humans, and waits for humans to interact with it. Detailed explanation of the setup and results of the human-robot interaction is presented in section 4.2.

3.6 Hypothesis

We hypothesize that establishing communication between the two frameworks using the two mentioned methods of establishing module-to-module communication will be consistent. However, it will be more efficient for the second choice of only sending new chunks of data and rebuilding the IU network in the destination. As for holding the entire IU network, we hypothesize that the overall IU network will be consistent, preserved, and can be accessed from both processes with both IU store options, however, the network will be more efficient where IUs are stored natively in one of the processes.

CHAPTER 4: EVALUATIONS

4.1 Experiment 1: Systematic Evaluation of Consistency and Efficiency

As mentioned in the previous chapters, the goal of this thesis is to demonstrate a working robot-ready dialog system pipeline in a distributed architecture where we use two processes (PSI and ReTiCo) to perform their fair share of the workload helping each other with their unique features. In order to achieve that, we implemented the robot-ready dialog system pipeline showed in Figure 3.1, and we implemented three types of IU data store to achieve the capability of holding the entire IU network generated by the pipeline so that both processes can traverse the network when necessary. We evaluated the pipeline and the different implementations of the data store separately.

4.1.1 Evaluation of the IU Store

In order to facilitate the conservation of the entire IU Network, we constructed two main variations of the IU data store as previously shown in Figure 3.4. The first type of IU store is constructed by choosing one process (PSI) and implementing it natively in that process. The other process (ReTiCo) relies on ZeroMQ to query relevant information (e.g. requesting the Grounded-in IU of a particular IU) from the other process. We built two variations of the native IU store: one in C# since PSI itself is based off of the .NET environment, and another in native C++ to explore the efficiency of a relatively lower-level language.

The second variant of IU store is constructed as a shared data storage to which both the processes have access, and they can both query the IU store directly to retrieve relevant data. As shown in Figure 3.4, this IU store acts as a middleman being a standalone entity and holds the entire IU network generated by both the frameworks. We used the in-memory data structure store Redis for the implementation of this IU store.

As mentioned in section 3.3.6, regardless of the type and implementation strategy, all of our IU store implementations have the following main functions: InsertIU, RetrieveIU, GetPreviousIU, and GetGroundedInIU. All of these functionalities are stress-tested in terms of latency and IU delivery success rate. We tested our IU store by generating, inserting, and retrieving current, previous, and Grounded-in IUs of 5000 IUs, and averaging the latency and IU delivery success rate for 50 iterations for each variant. For the PSI counterpart, we generated 5000 dummy Speech Recognition and Incrementalized Speech Recognition IUs each connected to its respective previous and Grounded-in IUs, whereas for the ReTiCo counterpart, the same amount of Dialog Act IUs (Rasa) and Dialog Decision IUs (PyOpenDial) were generated with the same requirements fulfilled for each iteration. For this test, I used the final completed network structure mentioned in section 3.4. Table 4.1 and Table 4.2 show the results for all of the variants of the IU store for PSI and ReTiCo respectively.

Function	Store Type	Latency	IU delivery success rate
RetrieveIU	Native (C#)	0.003ms	100%
	Native (C++)	0.098ms	100%
	Shared (Redis)	6.249ms	100%
GetPreviousIU	Native (C#)	0.007ms	100%
	Native (C++)	0.203ms	100%
	Shared (Redis)	12.513ms	100%
GetGroundedInIU	Native (C#)	0.005ms	100%
	Native (C++)	0.151ms	100%
	Shared (Redis)	10.498ms	100%

Table 4.1: Latency and IU delivery success rate of each function for each store type for PSI

Function	Store Type	Latency	IU delivery success rate
RetrieveIU	Native (C#)	15.661ms	100%
	Native (C++)	15.941ms	100%
	Shared (Redis)	0.098ms	100%
GetPreviousIU	Native (C#)	16.659ms	100%
	Native (C++)	16.744ms	100%
	Shared (Redis)	0.331ms	100%
GetGroundedInIU	Native (C#)	16.570ms	100%
	Native (C++)	16.605ms	100%
	Shared (Redis)	0.163ms	100%

Table 4.2: Latency and IU delivery success rate of each function for each store type for ReTiCo

As we can see, a native implementation (C#, C++) favors the native process with very low latency, but it is costly for the other process due to message passing and implementation constraints (e.g. continuous serialization/deserialization and traversing through the network to reach the destination process). On the other hand, the shared data storage implementation shows better performance than raw implementation that uses message passing as a means of communication. Redis performs better for ReTiCo than PSI because the shared data storage was facilitated in the same machine as ReTiCo. In order to determine whether that is true or if there are other factors to consider, I performed more stress-tests for Redis alone by moving Redis into a third machine and moving Redis to the machine where PSI is situated. Table 4.3 shows the result for Redis being situated into a third machine, whereas

Table 4.4 shows the result for Redis being situated in the same machine as PSI. These two tables suggest that the performance of the Shared IU store also depends of the connectivity. Specifically, the fact that the PSI machine was connected using Wi-Fi resulted in the increased latency for any communication with that particular machine. Additionally, we can see that the native C++ implementation was outperformed by the native C# implementation on every aspect due to its dependency on serialization and deserialization of objects. For that reason, we only used the native C# implementation and shared implementation by Redis in Experiment 2.

Function	Process	Latency	IU delivery success rate
RetrieveIU	PSI	6.658ms	100%
	ReTiCo	0.528ms	100%
GetPreviousIU	PSI	13.296ms	100%
	ReTico	1.918ms	100%
GetGroundedInIU	PSI	11.063ms	100%
	ReTiCo	1.099ms	100%

Table 4.3: Latency and IU delivery success rate of each Function for Shared (Redis) IU Store when it is situated in a third machine.

Function	Process	Latency	IU delivery success rate
RetrieveIU	PSI	0.988ms	100%
	ReTiCo	6.472ms	100%
GetPreviousIU	PSI	1.977ms	100%
	ReTico	22.216ms	100%
GetGroundedInIU	PSI	1.643ms	100%
	ReTiCo	11.094ms	100%

Table 4.4: Latency and IU delivery success rate of each function for Shared (Redis) IU Store when it is situated in the same machine as PSI.

Based on the data represented in Table 4.1 through Table 4.4, we can come to the following conclusions:

- If we build a distributed pipeline where one process more extensively traverses the network to get information than the other process, then implementing the IU store natively in that process will enable that process to access the IU store information with very low latency. In other words, a native implementation is preferred in that scenario.

- The performance has obvious dependency on the connectivity as well. An enclosed and wired connection ensures better communication for the IU store than a wireless connection.

- Although the latency for a Shared (Redis) IU store is not as low as a native store for the native process, a shared implementation is preferred when the distributed processes are using the IU store equally or almost equally.

4.1.2 Further Analysis of the Pipeline

Since the pipeline is distributed, we created unit tests for module-to-module communications where the modules are situated in different processes to ensure that IUs from the source modules produces the right IUs in the destination module. For example, speech recognition IUs generated in PSI Speech Recognition modules must produce the correct Rasa NLU IUs in ReTiCo as shown in Figure 4.1. The unit tests evaluated both the approaches mentioned in section 3.3.5. Both the approaches of sending reference to the entire SLL network and only sending the newly generated IUs and rebuilding the network in the destination module are stable, consistent, and preserve the network. However, during our rigorous testing, we found out that the first approach is not always stable for modules that deal with sensor data (e.g. microphone or camera). Since the communication between the two processes are done using ZeroMQ, sending the total reference to the entire SLL over and over from a source that is very fast and exhaustive (e.g. microphone) makes the network unstable. Due to that fact, the remaining evaluations were performed using only the second approach of sending the newly generated IUs and rebuilding it in the destination module.

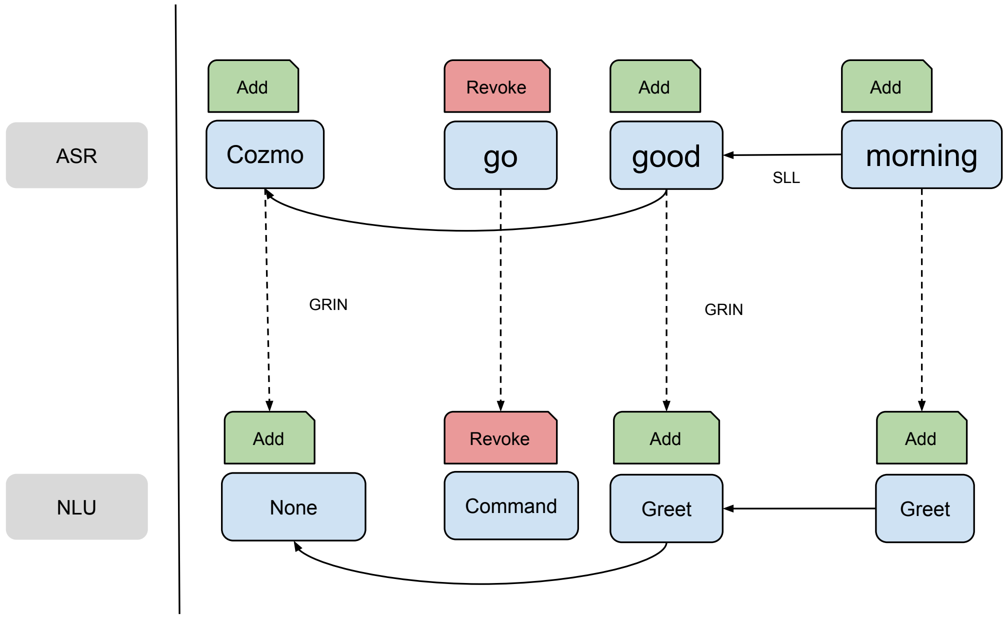

Figure 4.1: NLU creating appropriate IUs corresponding to ASR IUs.

When it comes to efficiency, we recorded the average time it takes for an IU to be generated starting from the moment the input IUs to that respective module is created. This information is illustrated in Table 4.5. For example, this table shows the time it takes for a Rasa NLU IU in ReTiCo to be created from the generation time of its input IUs (Speech Recognition IUs in PSI). We can see that modules that are in separate processes have higher latency in terms of IU generation in the destination module due to its dependency on message passing and serialization tasks between the two processes. Moreover, the PyOpenDial module uses the data store most extensively by calling the GetGroundedInIU and GetPreviousIU functions, since it needs information about several types of IUs (e.g. Speech Recognition, Object Detection, Natural Language Understanding). Since we proceeded with the structure mentioned in section 3.4, we can see the obvious advantage of implementing a shared data storage since Redis returns necessary data to PyOpenDial (which is part of ReTiCo machine in our implementation) in 95ms where the native store has an average latency of 301ms.

Table 4.5: IU generation latency at different modules.

Store Type	Rasa NLU IU	Keras Feature Extraction IU	PyOpenDial DM IU
Redis	0.613s	0.851s	0.106s
Native	0.581s	0.843s	0.286s

4.2 Experiment 2: Live, Interactive Study with Human Participants

Although we have our numerical and constructional results that demonstrate which methods are preferable over others based on IU data storage creation and IU module network construction, it is important to know about their impact on an actual dialog system setting where humans interact with the system. In order to evaluate this effect, we conducted a live experiment with users using the Cozmo robot in a setting as shown in Figure 3.5. Moreover, for the live interactive study with humans, I decided to stay with the setup mentioned in section 3.4 to get more diverse results from two IU store implementations, one of which is implemented natively with a non-wired connectivity away from the process that would utilize the store the most and the other one implemented in a shared structure in the same machine as the process that would utilize the store the most.

4.2.1 Participant Recruitment and Study Setting

We recruited fourteen study participants to interact with the Cozmo robot twice, each time being a ten-minute period over the course of a single session. Following each ten-minute interaction, the participants were asked to fill out a questionnaire that contained all the questions from the Godspeed Questionnaire (found in the Appendix) [39] along with additional questions related to our particular format of task setting. The additional questions are as follows:

- How much did the robot respond to your request/command?
- How much did the robot respond correctly to your request/command?
- How many times did the robot correctly identify an object color?
- How many times were you able to help the robot make it to the desired object?
- Do you think the robot responded to you within reasonable amount of time?

The Godspeed Questionnaire is a Likert-scaled questionnaire with 24 questions ranging from negative to positive ratings of a robot's anthropomorphism, animacy, likeability, perceived intelligence, and perceived safety. The entire study takes approximately one hour for each participant. In exchange for their valuable time, the participants were paid eight U.S. dollars. The study participants were mostly university students recruited from the Department of Computer Science at Boise State University, although 28.5% of the participants are from different disciplines. While 57% of the participants are native speakers, the remaining participants are near native. Five of the participants are women and nine are men.

We deployed two versions of the IU store (Native (C#) and Shared (Redis)) to construct the pipeline shown in Figure 3.1 to be used by the robot in the two ten-minute sub-sessions where the robot had three main functionalities as follows:

- The users can greet the robot, and the robot greets back (e.g. Hello, Good morning, Good Bye).
- The users can command the robot and make it move around the task setting (e.g. Go forward, Turn left, Drive back, Stop, Keep going, etc.) as shown in Figure 3.5.
- Once the robot is near an object, users can ask questions about the color of the blocks and expect appropriate response (e.g. What color is the block?, Is it blue?).

The researcher remained present near the task setting to monitor the state of the robot, troubleshoot any problems that might arise, and answer any questions or queries the participants might have over the course of the interaction with the robot. The researcher was permitted to offer a constrained set of coaching tips to the participant during the interaction. Part of the study was observed with cameras, which recorded audio and video from the interaction. Following each interaction, the user moved to the researcher's seat to complete the related questionnaire on the researcher's laptop. Following the completion of both interactions and subsequent surveys, the participant is paid eight dollars and signs a form acknowledging receipt of payment.

4.2.2 Results

Figure 4.2 through Figure 4.6 shows the answer for the “task-setting” specific questions and we can see that there is a general pattern about the preference of store type. For the first four questions, although the majority of participants thought that the robot was working correctly for both types, a few participants reported that the robot response could be better for type native (C#). This means that those particular participants recognized the response time difference between the Shared storage implemented in the same machine and the Native storage implemented in the other process. For example, for the first question (as shown in Figure 4.2), 35.71% of the participants agreed that the robot responded to a command all the time for both types, 64.29% and 50% of the participants agreed that the robot responded to a command most of the time for Shared and Native store type respectively. However, 14.29% of the participants reported that the response of the robot for the Native type could be better. For the fifth “test-setting” specific question of whether the robot responded to the participants in a reasonable amount of time, 85.72% and 64.28% of the participants agreed with a positive answer for Shared and Native type respectively. However, 14.28% and 35.72% of the participants for Shared and Native type respectively agreed that the response time can be improved. In addition to that, the mean of each of the “task-specific” questions for each type of IU Store is shown in Figure 4.7 and Table 4.6 which clearly suggest that although both types have positive means, the Shared IU store implemented in the same machine is desirable in terms of the performance of the robot.

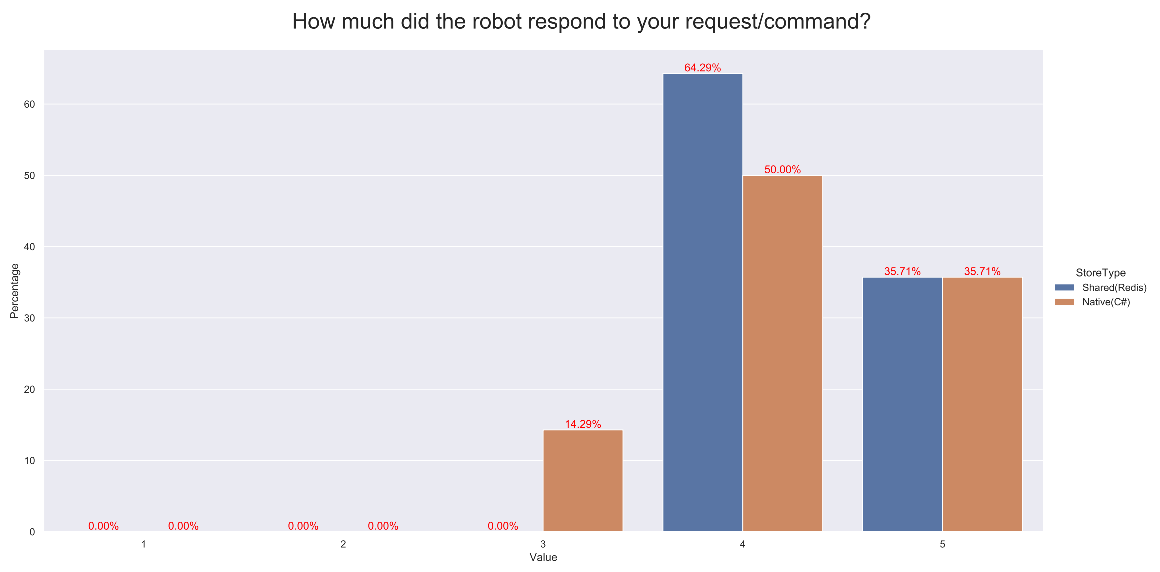

Figure 4.2: X-axis: Participant ratings from 1: Not at all to 5: All the time. Y-axis: the % of participants that selected those responses

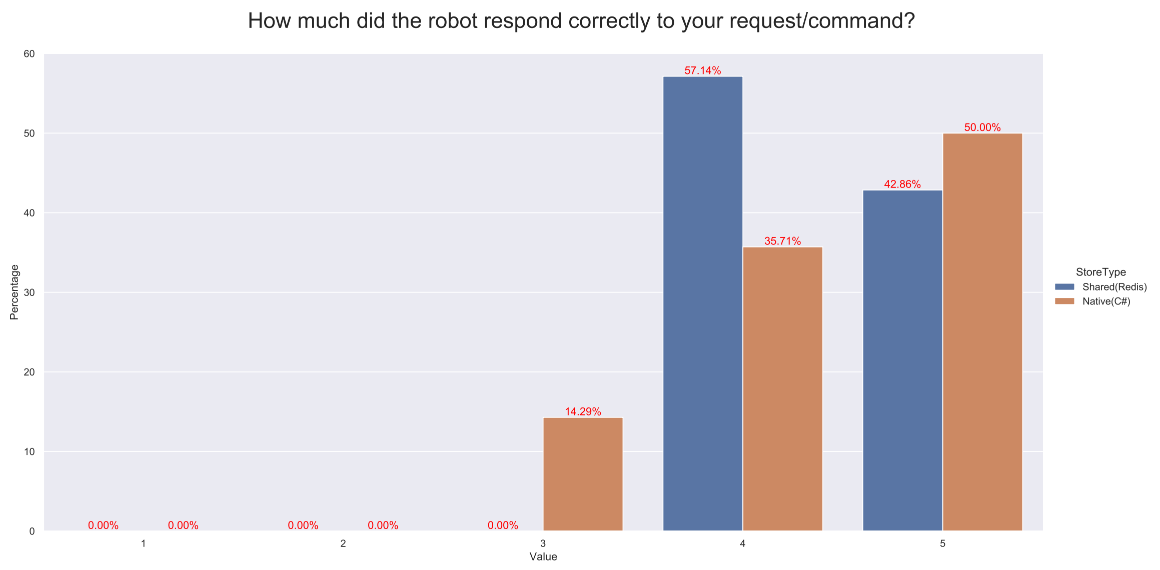

Figure 4.3: X-axis: Participant ratings from 1: Not at all to 5: All the time. Y-axis: the % of participants that selected those responses

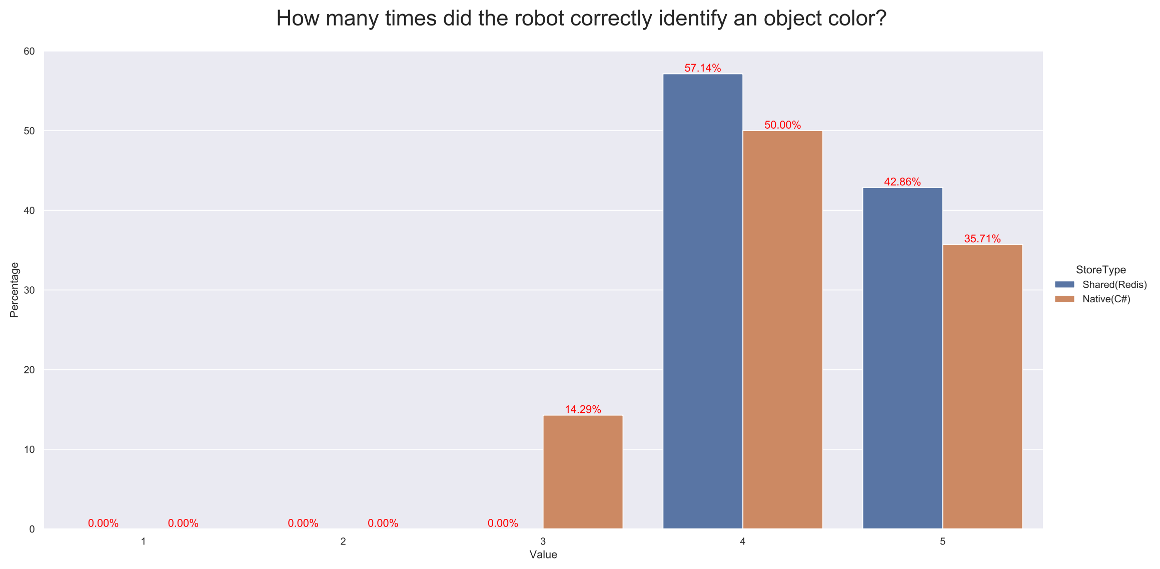

Figure 4.4: X-axis: Participant ratings from 1: Not at all to 5: All the time. Y-axis: the % of participants that selected those responses

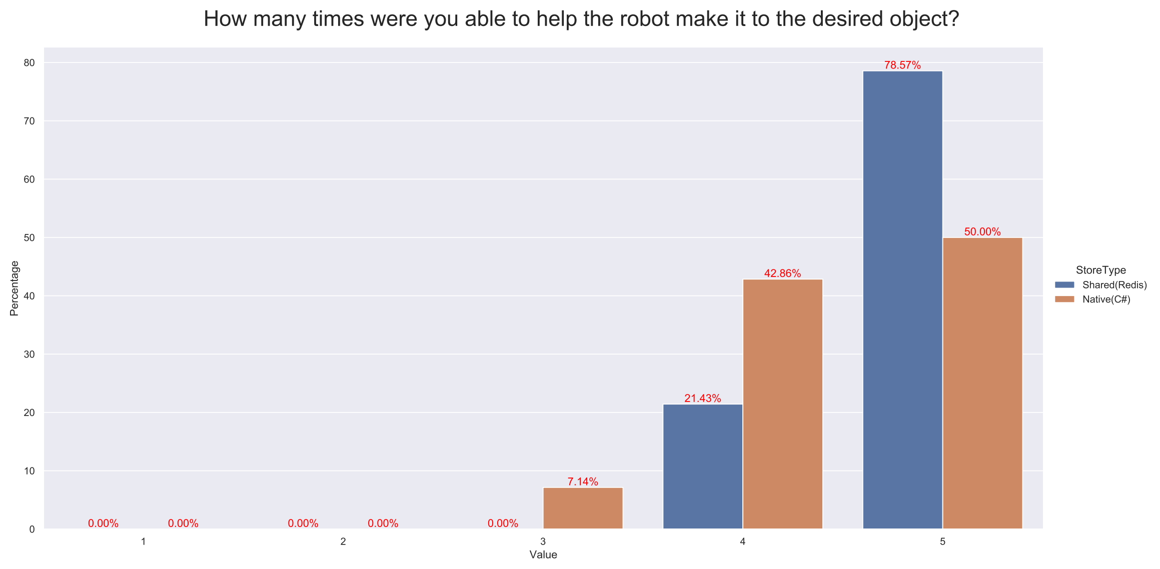

Figure 4.5: X-axis: Participant ratings from 1: Never to 5: Every time. Y-axis: the % of participants that selected those responses

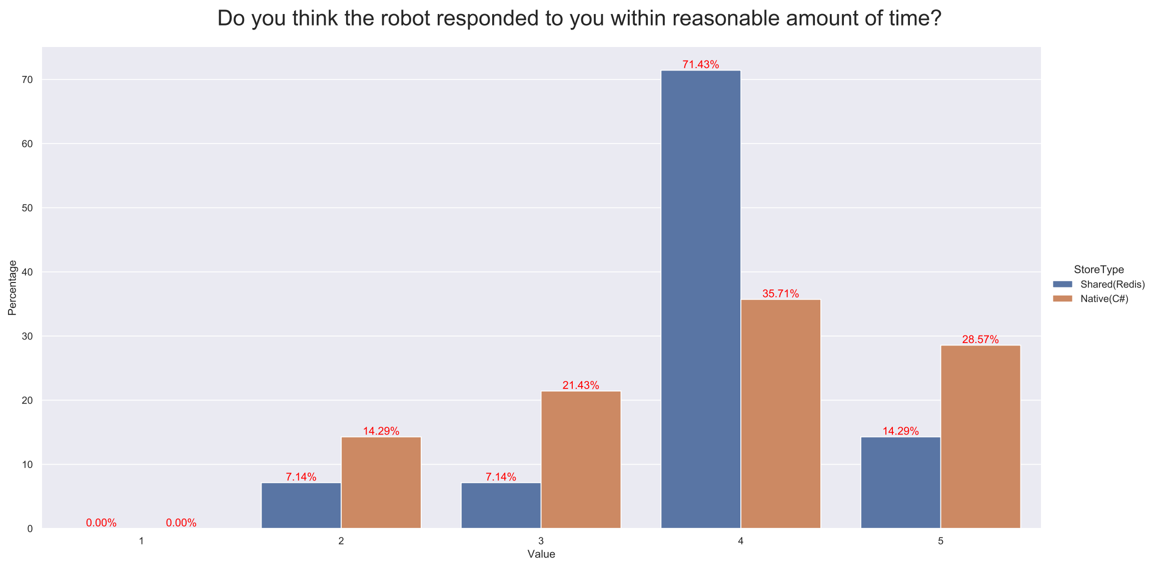

Figure 4.6: X-axis: Participant ratings from 1: Not reasonable at all to 5: Reasonable. Y-axis: the % of participants that selected those responses

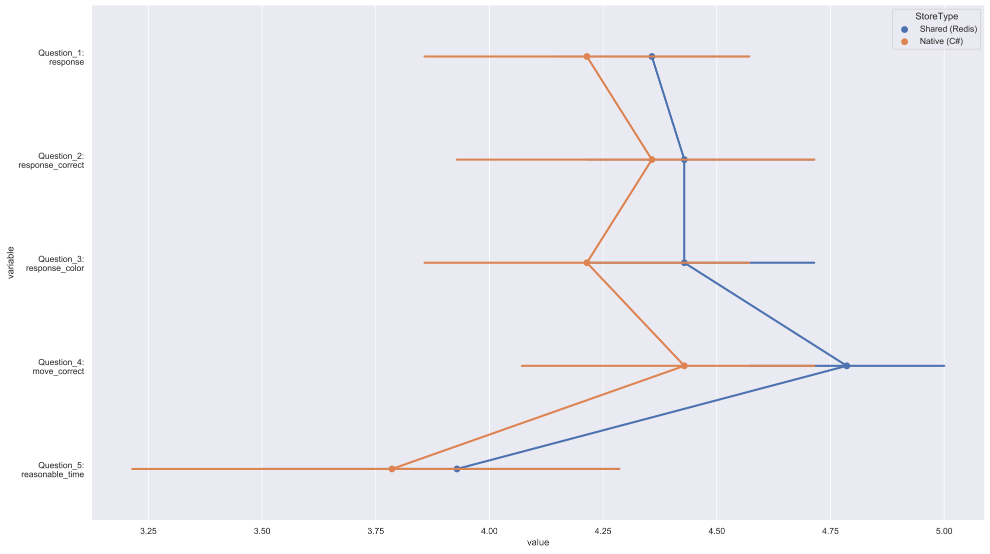

Figure 4.7: Mean participant response to each task-specific questions for each type of IU Store

Table 4.6: Mean of participant responses for task-specific questions.

Question	Redis	Native
How much did the robot respond to your request/command?	4.357143	4.214286
How much did the robot respond correctly to your request/command?	4.428571	4.357143
How many times did the robot correctly identify an object color?	4.428571	4.214286
How many times were you able to help the robot make it to the desired object?	4.785714	4.428571
Do you think the robot responded to you within reasonable amount of time?	3.928571	3.785714

Although this thesis focused more on building a consistent robot-ready distributed dialog system pipeline fulfilling all the requirements of a IU network, I have recorded important pieces of information that are related to the robot's anthropomorphism, animacy, likeability, and perceived intelligence. It appears that although the two IU storage implementations are different in terms of performance, the response to these questions from the Godspeed questionaire are not always proportional to the performance metrics. Figure 4.8 and Table 4.7 show the mean response to questions that preferred the Shared (Redis) IU Store whereas Figure 4.9 and Table 4.8 show the mean response to questions that preferred the Native (C#) IU Store. This indicates that although we chose two distinct types of IU store to implement with one having significant performance advantage over the other, the difference does not always directly reflect on the answer to these questions.

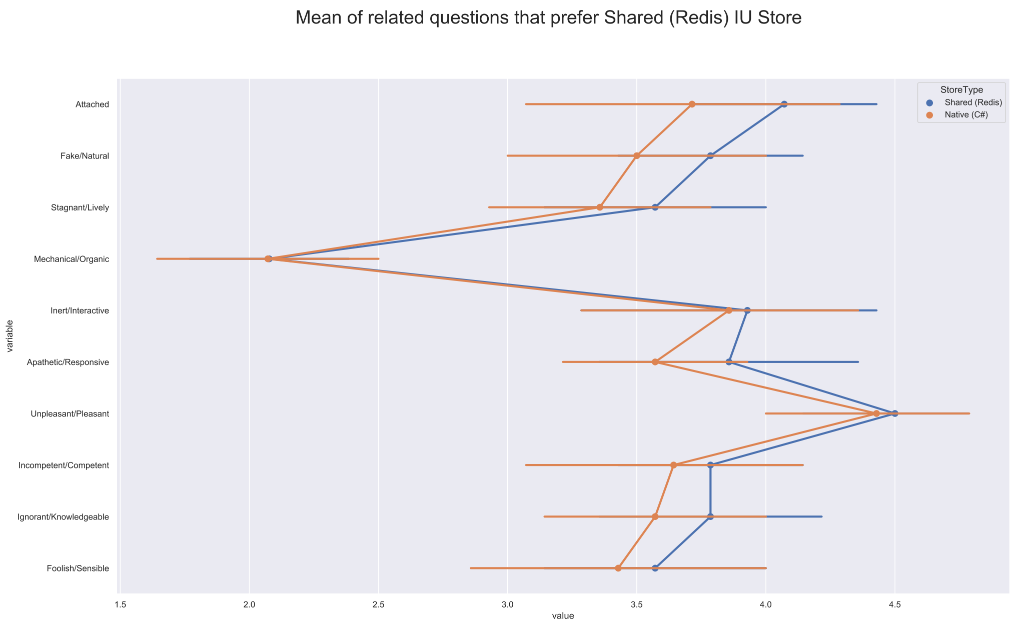

Figure 4.8: Mean participant response to questions related to the robot's anthropomorphism, animacy, likeability, and perceived intelligence that preferred the Shared (Redis) IU Store

Table 4.7: Mean of participant responses for questions that prefer Shared (Redis) IU Store.

Question	Redis	Native
How attached to the robot did you feel?	4.071429	3.714286
Fake/Normal	3.785714	3.500000
Stagnant/Lively	3.571429	3.357143
Mechanical/Organic	2.076923	2.071429
Inert/Interactive	3.928571	3.857143
Apathetic/Responsive	3.857143	3.571429
Unpleasant/Pleasant	4.500000	4.428571
Incompetent/Competent	3.785714	3.642857
Ignorant/Knowledgeable	3.785714	3.571429
Foolish/Sensible	3.571429	3.428571

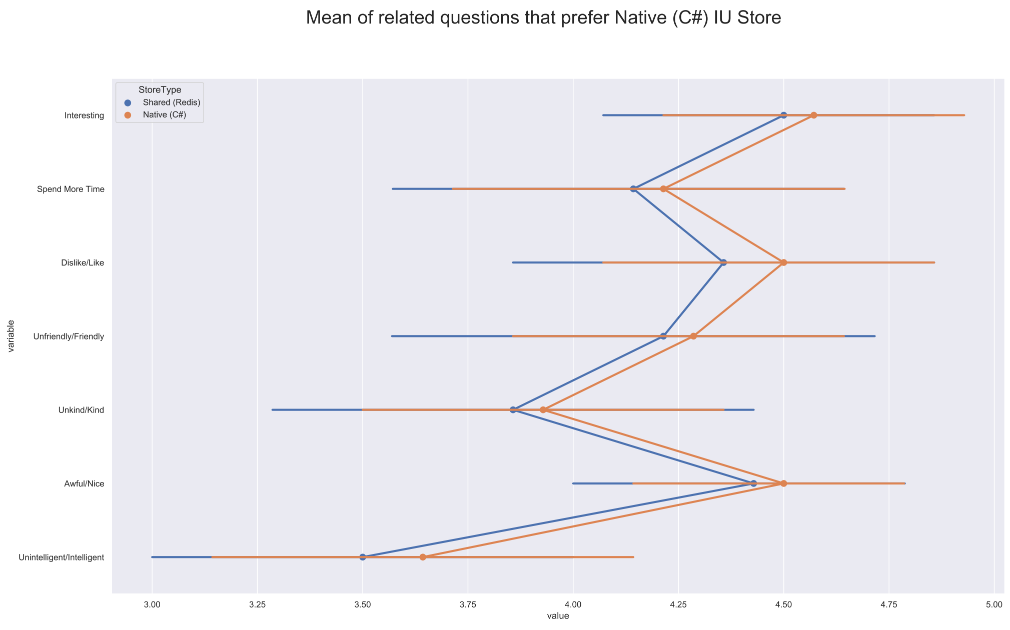

Figure 4.9: Mean participant response to questions related to the robot's anthropomorphism, animacy, likeability, and perceived intelligence that preferred the Native (C#) IU Store

Table 4.8: Mean of participant responses for questions that prefer Native (C#) IU Store.

Question	Redis	Native
How interesting was the robot to interact with?	4.500000	4.571429
Would you like to spend more time with the robot?	4.142857	4.214286
Dislike/Like	4.357143	4.500000
Unfriendly/Friendly	4.214286	4.285714
Unkind/Kind	3.857143	3.928571
Awful/Nice	4.428571	4.500000
Unintelligent/Intelligent	3.500000	3.642857

In order to understand the interdependence of the Godspeed questions with the performance of the IU stores even further, I have also evaluated the correlation between task-specific questions and other questions in the Godspeed questionnaire which is represented in Table 4.9, 4.10, and 4.11. We can see that the Shared (Redis) store has a high correlation between the reasonable response time and whether the robot is responsible/irresponsible, responsive/apathetic, and nice/awful. In other words, a better response time from the robot is treated as responsible, responsive, and nice. However, we can see that the Shared (Redis) store do not express as extensive correlation for these questions as the Native (C#) store. This demonstrates that an IU store with even only a small difference with a high performance IU store has a significant effect on how humans perceive the robot's anthropomorphism-related characteristics. In other words, a minor difference in performance results in human participants concerning the overall aura of the robot more. It affects how much the participants perceive the robot as natural, sensible, friendly, and how calm it made them feel. For example, there is a very high correlation between the robot responding correctly and being responsible/irresponsible which is not present for the Shared (Redis) store. This illustrates that even a small delay in performance changes the way human participants evaluate the robot. When the performance decreases, other factors gets highlighted more in deciding the overall perception. When the pipeline is very efficient, humans tend to overlook some characteristics being pleased with the efficient performance and being a little apathetic about the anthropomorphism-related characteristics. But when it is even slightly delayed, humans subconsciously get more judgemental to the robot's characteristics in relation to the performance. Moreover, these results verify the results in Novikova et al. [40] and Plane et al. [41]

that showed how robot movements affect human perceptions of those robots; in this case, any processing delay in understanding and responding to spoken utterances was perceived negatively.

Table 4.9: Correlation between task-setting related questions and other Godspeed questions that have a higher correlation than 0.45 for Shared IU store.

Question1	Question2	Correlation (Shared)	Correlation (Native)
response_color	lively_stagnant	0.452267	0.251985
response_color	pleasant_unpleasant	0.460566	0.350592
response_color	responsible_irresponsible	0.559793	0.269862
reasonable_time	interesting	0.485529	0.456488
reasonable_time	spend_more_time	0.527589	0.498726
reasonable_time	natural_fake	0.570420	0.350211
reasonable_time	moveElegantly_moveRigidly	0.563213	0.195482
reasonable_time	responsive_apathetic	0.561328	0.456488
reasonable_time	like_dislike	0.607751	0.556567
reasonable_time	friendly_unfriendly	0.551019	0.696761
reasonable_time	kind_unkind	0.465491	0.222393
reasonable_time	nice_awful	0.671647	0.281336

Table 4.10: Correlation between task-setting related questions and other Godspeed questions that have a higher correlation than 0.45 for Native IU Store Table 1.

Question1	Question2	Correlation (Shared)	Correlation (Native)
response	competent_incompetent	0.399667	0.515672
response_correct	lively_stagnant	0.276385	0.516888
response_correct	interactive_inert	0.199681	0.473950
response_correct	friendly_unfriendly	0.216085	0.696984
response_correct	competent_incompetent	0.053376	0.647699
response_correct	knowledgable_ignorant	0.047946	0.502320
response_correct	responsible_irresponsible	0.353553	0.759972
response_correct	sensible_foolish	0.276385	0.460385
response_correct	end_relaxed_anxious	-0.345582	0.534919
response_correct	begin_calm_agitated	-0.172345	0.516888
response_correct	end_calm_agitated	0.000000	0.662004
response_color	attracted	-0.087932	0.469200
response_color	interesting	0.000000	0.478130
response_color	interactive_inert	0.199681	0.474292
response_color	responsive_apathetic	0.389249	0.478130
response_color	like_dislike	0.138233	0.450376
response_color	friendly_unfriendly	0.342134	0.552106
response_color	begin_relaxed_anxious	0.299504	0.495769
response_color	begin_calm_agitated	0.229794	0.774619

Table 4.11: Correlation between task-setting related questions and other Godspeed questions that have a higher correlation than 0.45 for Native IU Store Table 2.

Question1	Question2	Correlation (Shared)	Correlation (Native)
response_color	end_calm_agitated	0.394405	0.559314
response_color	begin_interested_bored	0.228218	0.514174
move_correct	natural_fake	-0.166070	0.632817
move_correct	conscious_unconscious	-0.339945	0.597284
move_correct	lifelike_artificial	0.028239	0.453423
move_correct	friendly_unfriendly	0.097728	0.473849
move_correct	pleasant_unpleasant	-0.138866	0.479234
move_correct	nice_awful	0.060606	0.549031
move_correct	competent_incompetent	-0.144841	0.455860
move_correct	responsible_irresponsible	0.035533	0.817689
move_correct	sensible_foolish	-0.272727	0.593171
reasonable_time	attracted	0.298969	0.587700
reasonable_time	humanlike_machinelike	0.189629	0.523799
reasonable_time	interactive_inert	0.386281	0.539582
reasonable_time	pleasant_unpleasant	0.404983	0.454304
reasonable_time	sensible_foolish	0.318148	0.489525
reasonable_time	end_relaxed_anxious	0.040517	0.490710
reasonable_time	end_calm_agitated	0.208084	0.647270

CHAPTER 5: CONCLUSION

In this thesis, I attempted to bring all the key characteristics a standard dialog system should have into one place facilitating the preservation of entire IU network without breaking the connection, and I have accomplished that in two main different ways and compared their strengths and differences. Although I have used two main versions of my implementation in the final experiment, I have shown ways to maximize the performance of each of the different versions and where one can be more useful over the other. In terms of connecting modules distributed between processes, we have seen that the second approach of only sending new chunks of data and rebuilding the IU network in the destination is the better choice. In addition to the main objective of building a complete robot-ready dialog system pipeline, I have also demonstrated that meaningful stream fusion is feasible between distributed streams of data, and multiple NLU modules can be integrated in the same pipeline to work together. Furthermore my work demonstrates the effect of implementational changes on how humans perceive a robot's anthropomorphism-related characteristics, and that can be improved with a more efficient implementation. I believe this thesis will inspire the construction of larger and more complex implementations, and that will lead to further research in this area.

5.1 Limitations and Future Work

As mentioned above we have not explored all the possible combinations that would maximize the performance of the two types of IU Store. Instead, I experimented with one IU Store with obvious advantages above the other in order to realize the effect on overall response about the robot's anthropomorphism, animacy, likeability, and perceived intelligence. A live study on the two stores maximizing the performance of both variants should reveal more information about their application. In addition to that, comparing the complete distributed robot-ready dialog system pipeline with a non-distributed analogous pipeline will also reveal insights about its performance.

In my experiment, I relied on message passing techniques based on JSON in order to establish connections between modules situated in difference processes. Using other types of serialized structured data (e.g. Protocol Buffers from Google and Thrift from Facebook) that have faster serialization time than JSON should increase the performance as well.

REFERENCES

[1] D. Spiliotopoulos, I. Androutsopoulos, and C. D. Spyropoulos, “Human-robot interaction based on spoken natural language dialogue,” in *Proceedings of the European workshop on service and humanoid robots*, 2001, pp. 25–27.

[2] T. Fong, C. Thorpe, and C. Baur, “Collaboration, dialogue and human-robot interaction, 10th international sumposium of robotics research (lorne, victoria, australia),” in *Proceedings of the 10th International Symposium of Robotics Research*, 2001.

[3] D. Bohus, S. Andrist, and M. Jalobeanu, “Rapid development of multimodal interactive systems: a demonstration of platform for situated intelligence,” in *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, 2017, pp. 493–494.

[4] T. Michael and S. Möller, “Retico: An open-source framework for modeling real-time conversations in spoken dialogue systems,” *Studientexte zur Sprachkommunikation: Elektronische Sprachsignalverarbeitung 2019*, pp. 134–140, 2019.

[5] C. Kennington, D. Moro, L. Marchand, J. Carns, and D. McNeill, “rrsds: Towards a robot-ready spoken dialogue system,” in *Proceedings of the 21th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 2020, pp. 132–135.

[6] A. Singh, V. Goswami, V. Natarajan, Y. Jiang, X. Chen, M. Shah, M. Rohrbach, D. Batra, and D. Parikh, "Mmf: A multimodal framework for vision and language research," https://github.com/facebookresearch/mmf, 2020.

[7] D. Schlangen and G. Skantze, "A general, abstract model of incremental dialogue processing," *Dialogue and Discourse*, vol. 2, no. 1, pp. 83–111, 2011. [Online]. Available: https://journals.linguisticsociety.org/elanguage/dad/article/download/361/361-2892-1-PB.pdf

[8] T. Baumann and D. Schlangen, "The inprotk 2012 release," in *NAACL-HLT Workshop on Future directions and needs in the Spoken Dialog Community: Tools and Data (SDCTD 2012)*, 2012, pp. 29–32.

[9] C. Kennington, T. Han, and D. Schlangen, "Temporal alignment using the incremental unit framework," in *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, 2017, pp. 297–301.

[10] L. W. J.M., *Speaking.* Cambridge, USA: MIR Press, 1989.

[11] C. Kennington, "Incrementally resolving references in order to identify visually present objects in a situated dialogue setting," 2016.

[12] C. Kennington, S. Kousidis, and D. Schlangen, "Inprotks: A toolkit for incremental situated processing," in *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, 2014, pp. 84–88.

[13] M. K. Tanenhaus, M. J. Spivey-Knowlton, K. M. Eberhard, and J. C. Sedivy, "Integration of visual and linguistic information in spoken language comprehension," *Science*, vol. 268, no. 5217, pp. 1632–1634, 1995.

[14] G. Aist, J. Allen, E. Campana, and C. G. Gallo, "Incremental understanding in human-computer dialogue and experimental evidence for advantages over non-incremental methods," *Decalog 2007*, p. 149, 2007.

[15] O. Buß and D. Schlangen, "Modelling sub-utterance phenomena in spoken dialogue systems," in *Proceedings of the 14th International Workshop on the Semantics and Pragmatics of Dialogue (Pozdial 2010)*, 2010.

[16] M. Slee, A. Agarwal, and M. Kwiatkowski, "Thrift: Scalable cross-language services implementation," *Facebook White Paper*, vol. 5, no. 8, 2007.

[17] A. S. TANENBAUM and M. VAN STEEN, "Processes," *Distributed Systems Principles and Paradigms,*, pp. 69–114, 2006.

[18] T. F. Bissyandé, F. Thung, D. Lo, L. Jiang, and L. Réveillère, "Popularity, interoperability, and impact of programming languages in 100,000 open source projects," in *2013 IEEE 37th annual computer software and applications conference.* IEEE, 2013, pp. 303–312.

[19] M. Grimmer, R. Schatz, C. Seaton, T. Würthinger, M. Luján, and H. Mössenböck, "Cross-language interoperability in a multi-language runtime," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 40, no. 2, pp. 1–43, 2018.

[20] D. Chisnall, "The challenge of cross-language interoperability," *Communications of the ACM*, vol. 56, no. 12, pp. 50–56, 2013.

[21] A. Verma, "Most popular programming languages for ma-

chine learning and data science," https://fossbytes.com/popular-top-programming-languages-machine-learning-data-science/, 2016.

[22] M. Wegiel and C. Krintz, "Cross-language, type-safe, and transparent object sharing for co-located managed runtimes," *ACM Sigplan Notices*, vol. 45, no. 10, pp. 223–240, 2010.

[23] M. Grimmer, C. Seaton, R. Schatz, T. Würthinger, and H. Mössenböck, "High-performance cross-language interoperability in a multi-language runtime," in *Proceedings of the 11th Symposium on Dynamic Languages*, 2015, pp. 78–90.

[24] M. Marge, S. Nogar, C. J. Hayes, S. M. Lukin, J. Bloecker, E. Holder, and C. Voss, "A research platform for multi-robot dialogue with humans," *arXiv preprint arXiv:1910.05624*, 2019.

[25] S. M. Lukin, F. Gervits, C. J. Hayes, A. Leuski, P. Moolchandani, J. G. Rogers III, C. S. Amaro, M. Marge, C. R. Voss, and D. Traum, "Scoutbot: A dialogue system for collaborative navigation," *arXiv preprint arXiv:1807.08074*, 2018.

[26] Y. Wang, R. C. Murray, H. Bao, and C. Rose, "Agent-based dynamic collaboration support in a smart office space," in *Proceedings of the 21th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 2020, pp. 257–260.

[27] D. Adamson, G. Dyke, H. Jang, and C. P. Rosé, "Towards an agile approach to adapting dynamic collaboration support to student needs," *International Journal of Artificial Intelligence in Education*, vol. 24, no. 1, pp. 92–124, 2014.

[28] S. Kousidis, C. Kennington, T. Baumann, H. Buschmeier, S. Kopp, and D. Schlangen, “A multimodal in-car dialogue system that tracks the driver’s attention,” in *Proceedings of the 16th international conference on multimodal interaction*, 2014, pp. 26–33.

[29] R. Math, A. Mahr, M. M. Moniri, and C. Müller, “Opends: A new open-source driving simulator for research,” *GMM-Fachbericht-AmE 2013*, vol. 2, 2013.

[30] J. Wienke and S. Wrede, “A middleware for collaborative research in experimental robotics,” in *2011 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2011, pp. 1183–1190.

[31] B. Carlmeyer, D. Schlangen, and B. Wrede, “Towards closed feedback loops in hri: Integrating inprotk and pamini,” in *Proceedings of the 2014 Workshop on Multimodal, Multi-Party, Real-World Human-Robot Interaction*, 2014, pp. 1–6.

[32] J. Peltason and B. Wrede, “Pamini: A framework for assembling mixed-initiative human-robot interaction from generic interaction patterns,” in *Proceedings of the SIGDIAL 2010 Conference*, 2010, pp. 229–232.

[33] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn,” in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2980–2988.

[34] Y. Jang, J. Lee, J. Park, K.-H. Lee, P. Lison, and K.-E. Kim, “PyOpenDial: A python-based domain-independent toolkit for developing spoken dialogue systems with probabilistic rules,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*

(EMNLP-IJCNLP): System Demonstrations. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 187–192. [Online]. Available: https://www.aclweb.org/anthology/D19-3032

[35] C. Kennington and D. Schlangen, "Simple learning and compositional application of perceptually grounded word meanings for incremental reference resolution," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers).* Beijing, China: Association for Computational Linguistics, Jul. 2015, pp. 292–301. [Online]. Available: https://www.aclweb.org/anthology/P15-1029

[36] J. L. Carlson, *Redis in action.* Manning Publications Co., 2013.

[37] S. Sun, J. Gong, A. Y. Zomaya, and A. Wu, "A distributed incremental information acquisition model for large-scale text data," *Cluster Computing*, vol. 22, no. 1, pp. 2383–2394, 2019.

[38] A. Venkataraman and K. K. Jagadeesha, "Evaluation of inter-process communication mechanisms," *Architecture*, vol. 86, p. 64, 2015.

[39] C. Bartneck, D. Kulić, E. Croft, and S. Zoghbi, "Measurement instruments for the anthropomorphism, animacy, likeability, perceived intelligence, and perceived safety of robots," *International journal of social robotics*, vol. 1, no. 1, pp. 71–81, 2009.

[40] J. Novikova, G. Ren, and L. Watts, "It's not the way you look, it's how you move:

validating a general scheme for robot affective behaviour," in *IFIP Conference on Human-Computer Interaction*. Springer, 2015, pp. 239–258.

[41] S. Plane, A. Marvasti, T. Egan, and C. Kennington, "Predicting perceived age: Both language ability and appearance are important," in *Proceedings of the 19th Annual SIGdial Meeting on Discourse and Dialogue*, 2018, pp. 130–139.

APPENDIX A: ANSWERS TO THE QUESTIONS IN GODSPEED QUESTIONAIRE

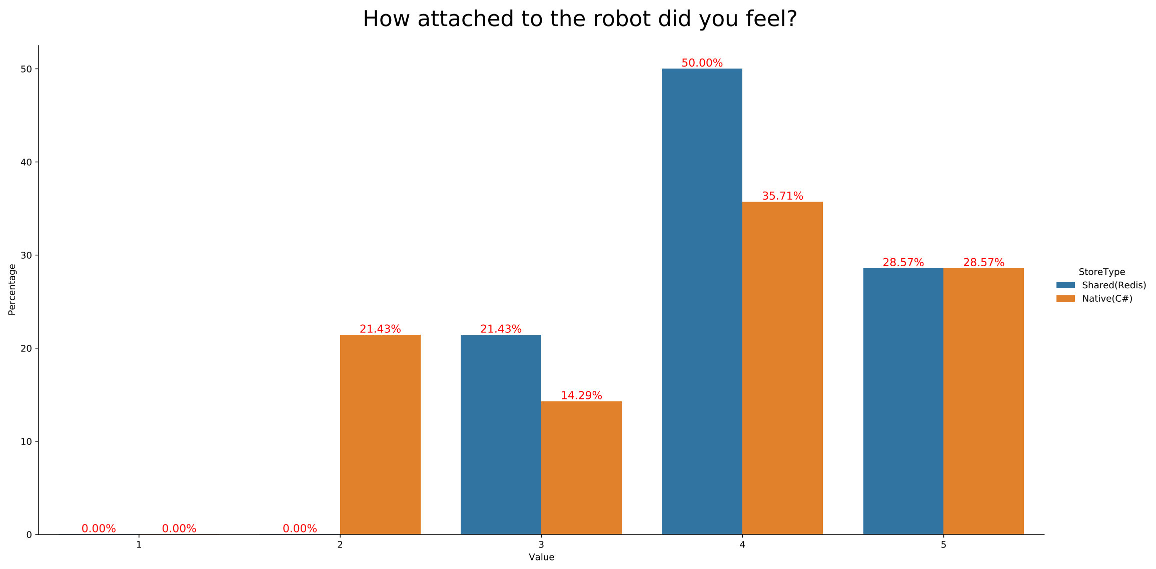

Figure A.1: X-axis: Participant ratings from 1: Not at all to 5: Very. Y-axis: the % of participants that selected those responses

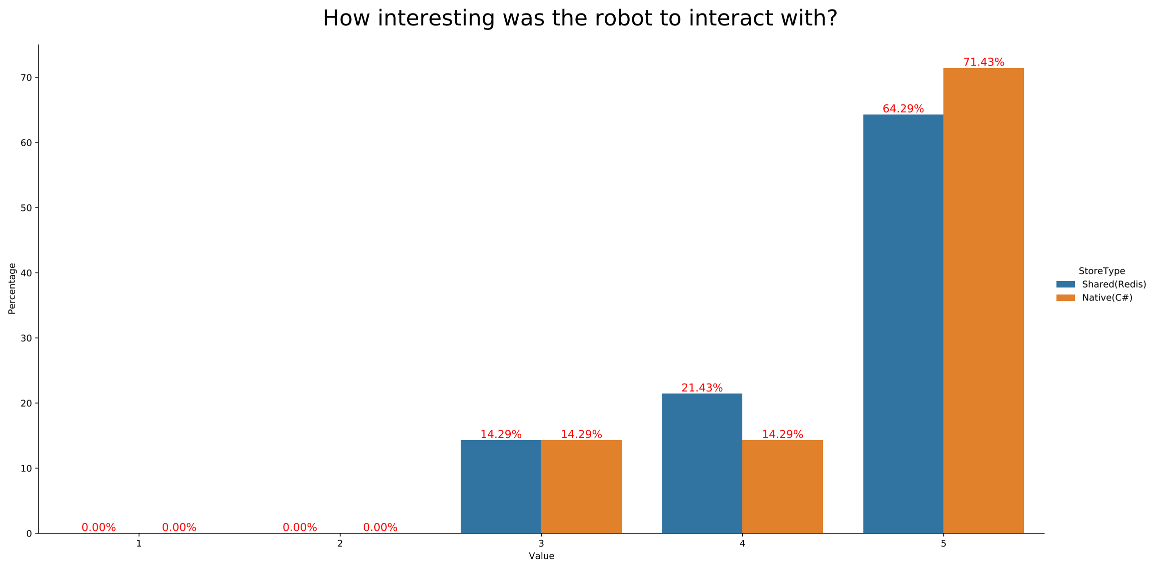

Figure A.2: X-axis: Participant ratings from 1: Not at all to 5: Very. Y-axis: the % of participants that selected those responses

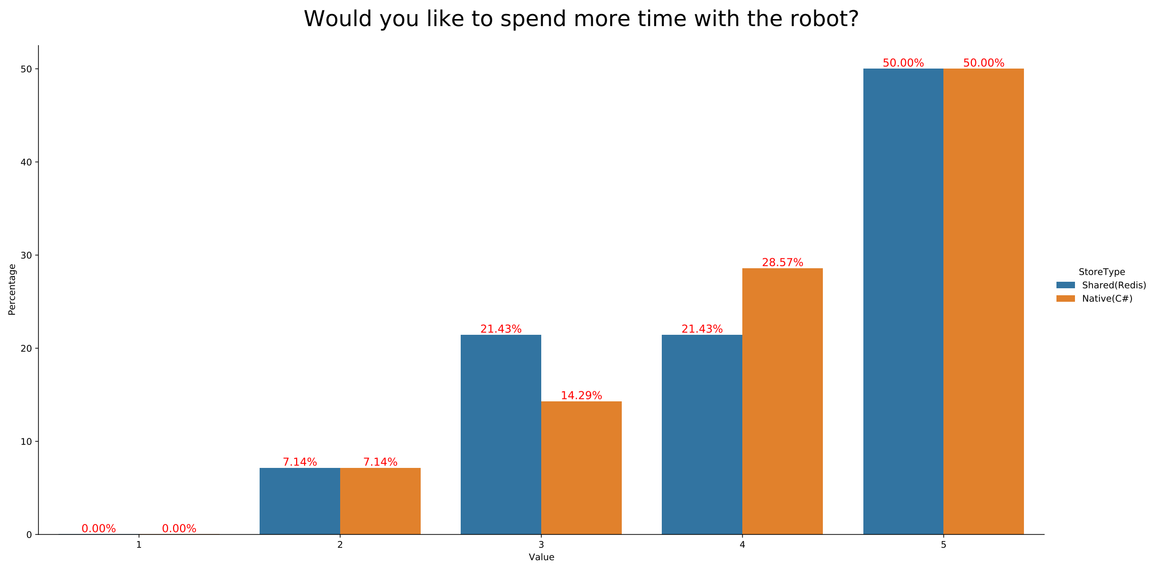

Figure A.3: X-axis: Participant ratings from 1: Not at all to 5: Very much. Y-axis: the % of participants that selected those responses

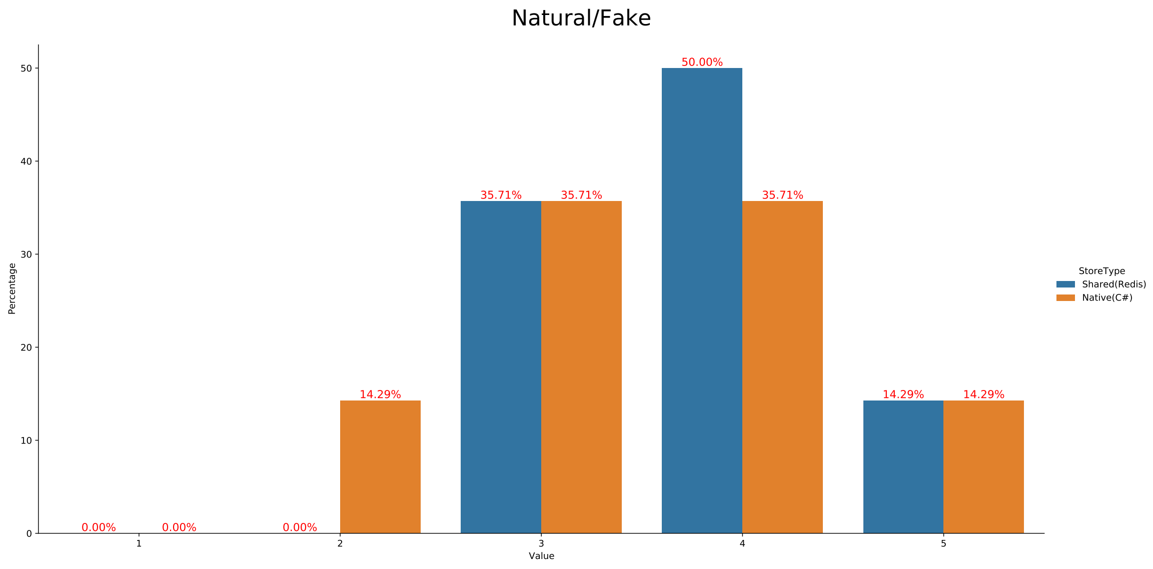

Figure A.4: X-axis: Participant ratings from 1: Fake to 5: Natural. Y-axis: the % of participants that selected those responses

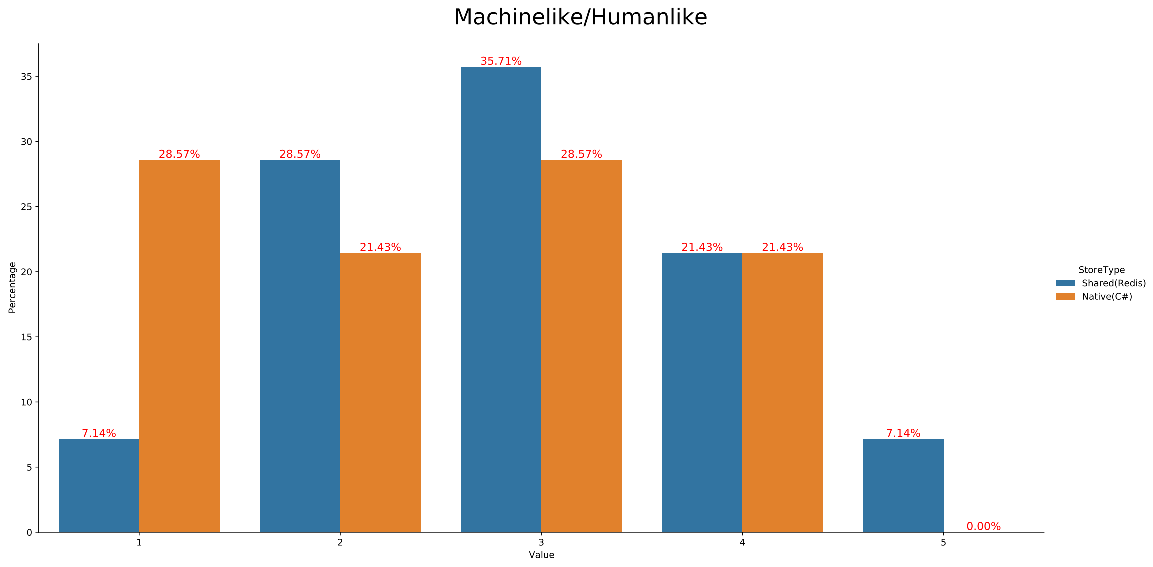

Figure A.5: X-axis: Participant ratings from 1: Machinelike to 5: Humanlike. Y-axis: the % of participants that selected those responses

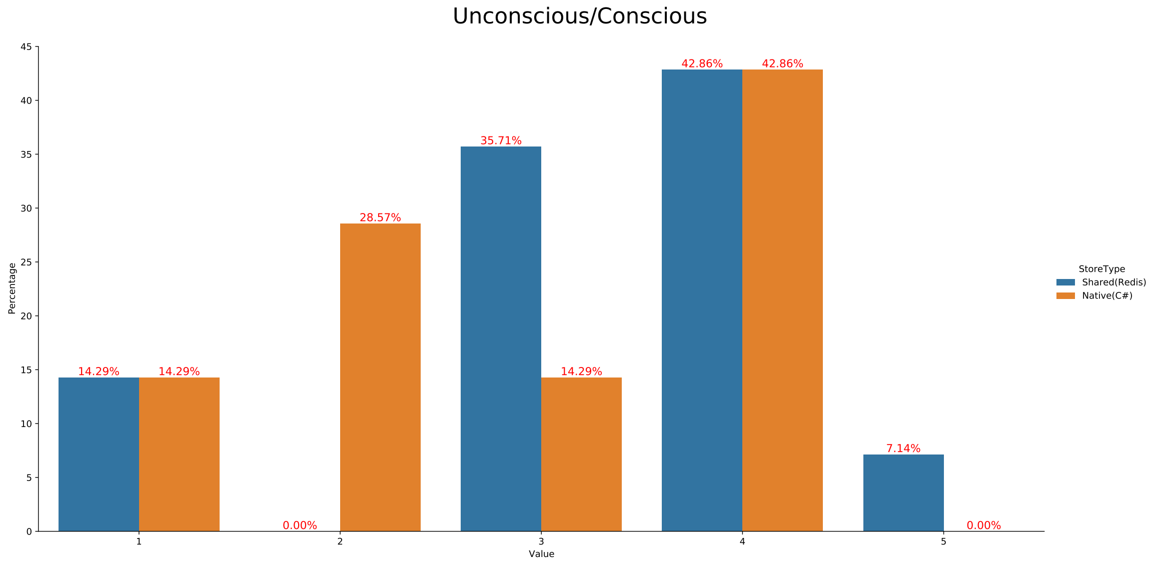

Figure A.6: X-axis: Participant ratings from 1: Unconscious to 5: Conscious. Y-axis: the % of participants that selected those responses

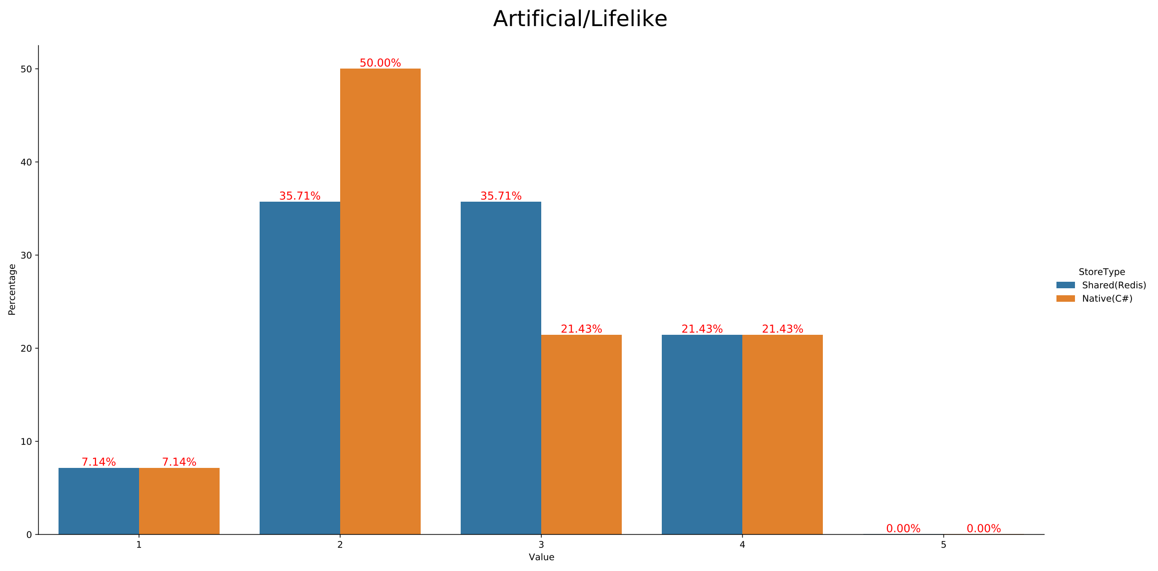

Figure A.7: X-axis: Participant ratings from 1: Artificial to 5: Lifelike. Y-axis: the % of participants that selected those responses

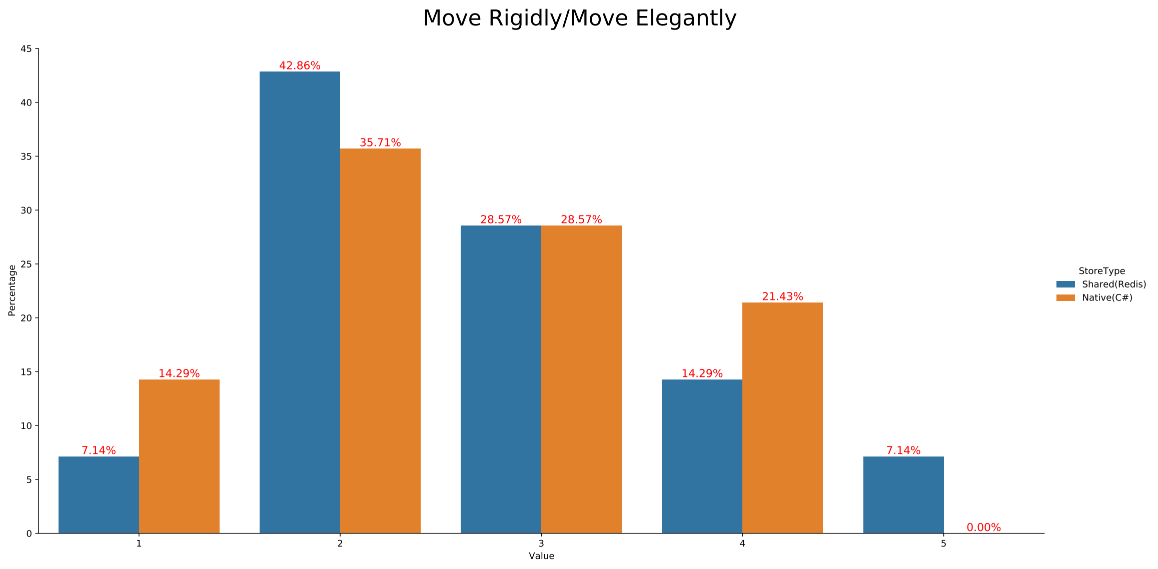

Figure A.8: X-axis: Participant ratings from 1: Moving Rigidly to 5: Moving Elegantly. Y-axis: the % of participants that selected those responses

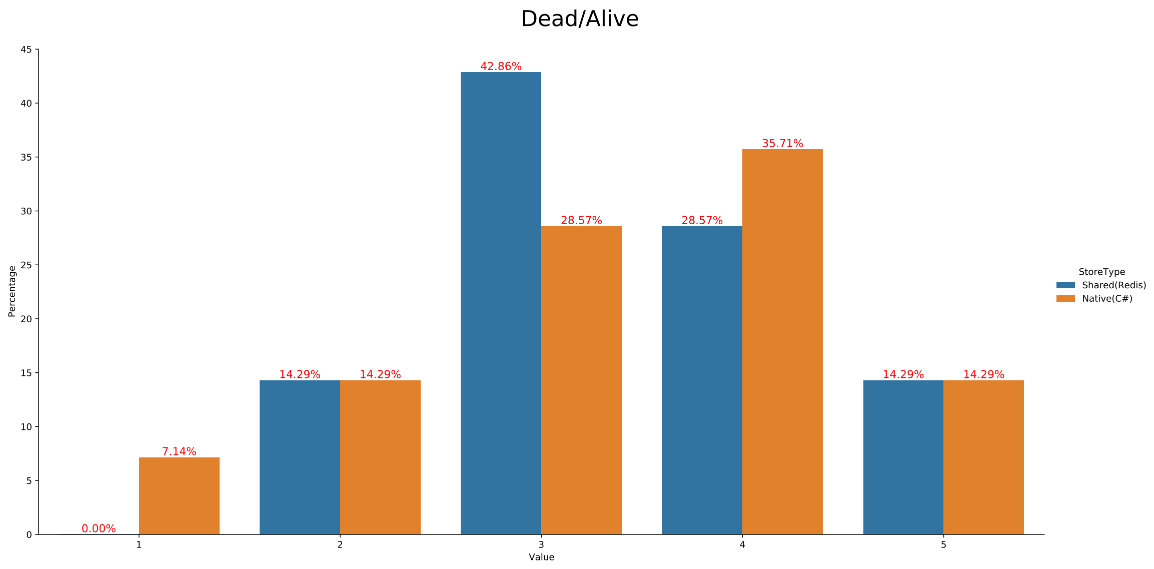

Figure A.9: X-axis: Participant ratings from 1: Dead to 5: Alive. Y-axis: the % of participants that selected those responses

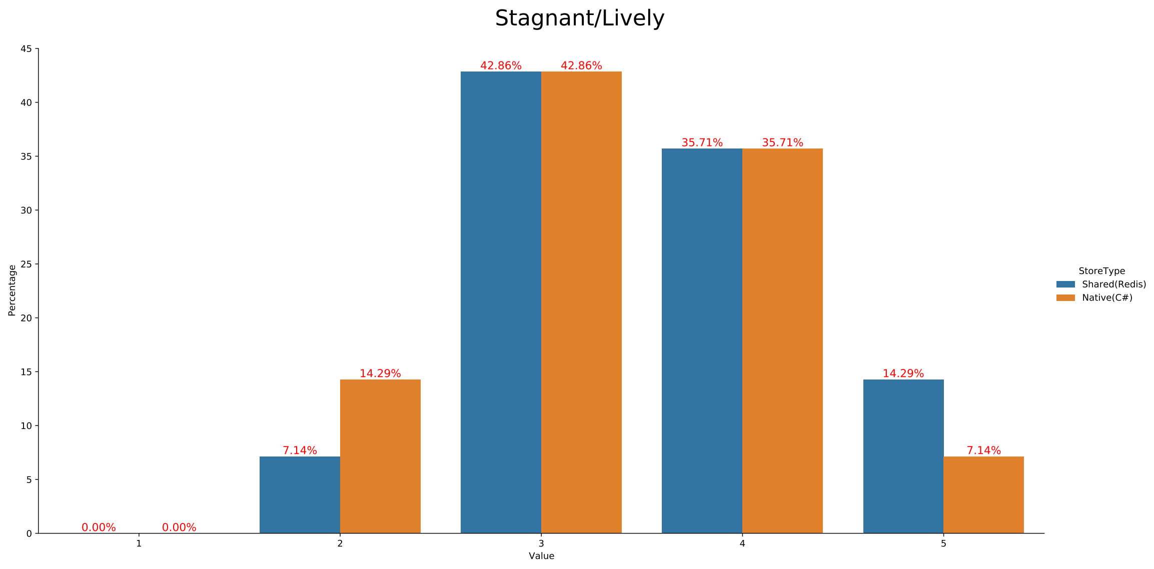

Figure A.10: X-axis: Participant ratings from 1: Stagnant to 5: Lively. Y-axis: the % of participants that selected those responses

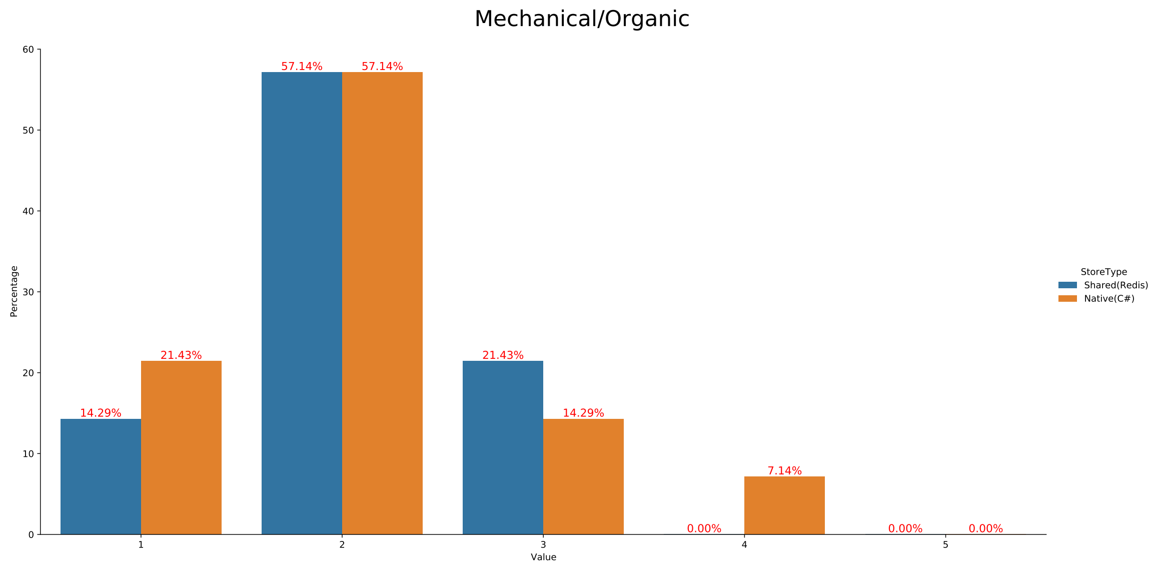

Figure A.11: X-axis: Participant ratings from 1: Mechanical to 5: Organic. Y-axis: the % of participants that selected those responses

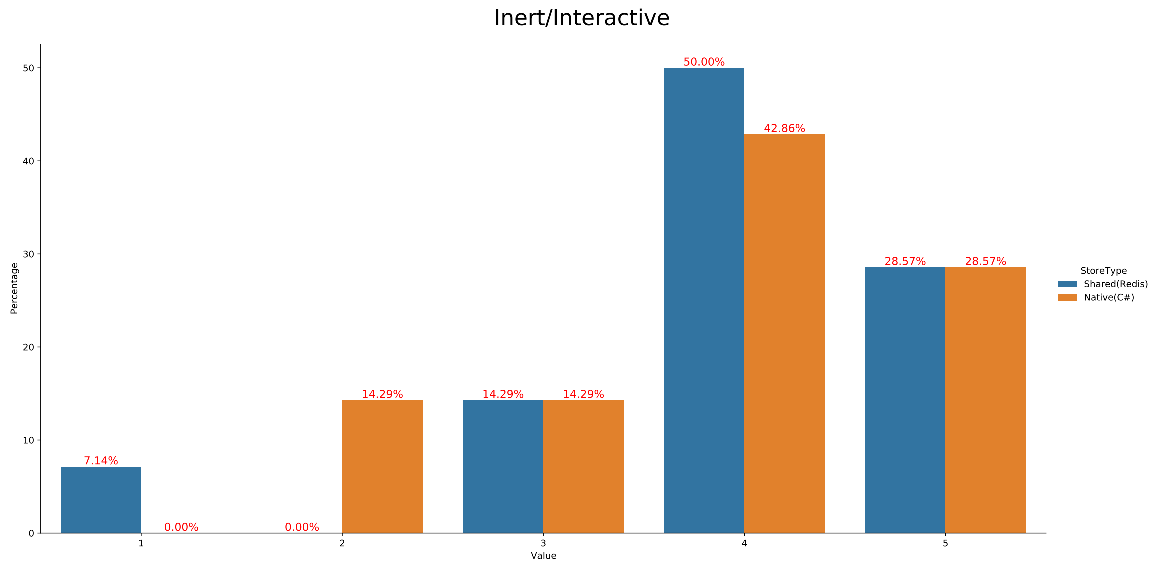

Figure A.12: X-axis: Participant ratings from 1: Inert to 5: Interactive. Y-axis: the % of participants that selected those responses

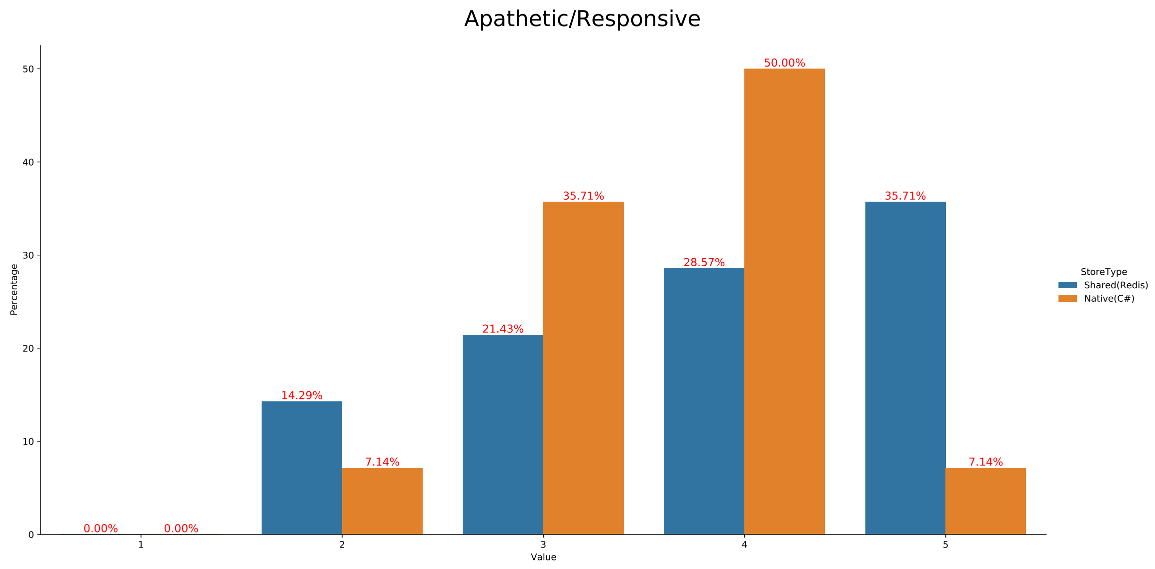

Figure A.13: X-axis: Participant ratings from 1: Apathetic to 5: Responsive. Y-axis: the % of participants that selected those responses

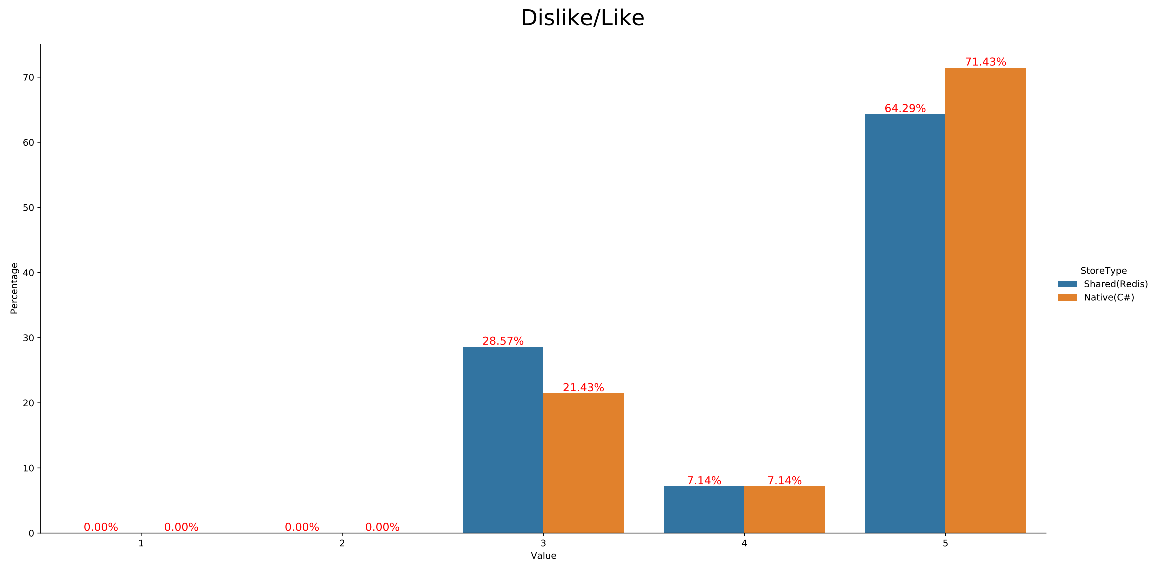

Figure A.14: X-axis: Participant ratings from 1: Dislike to 5: Like. Y-axis: the % of participants that selected those responses

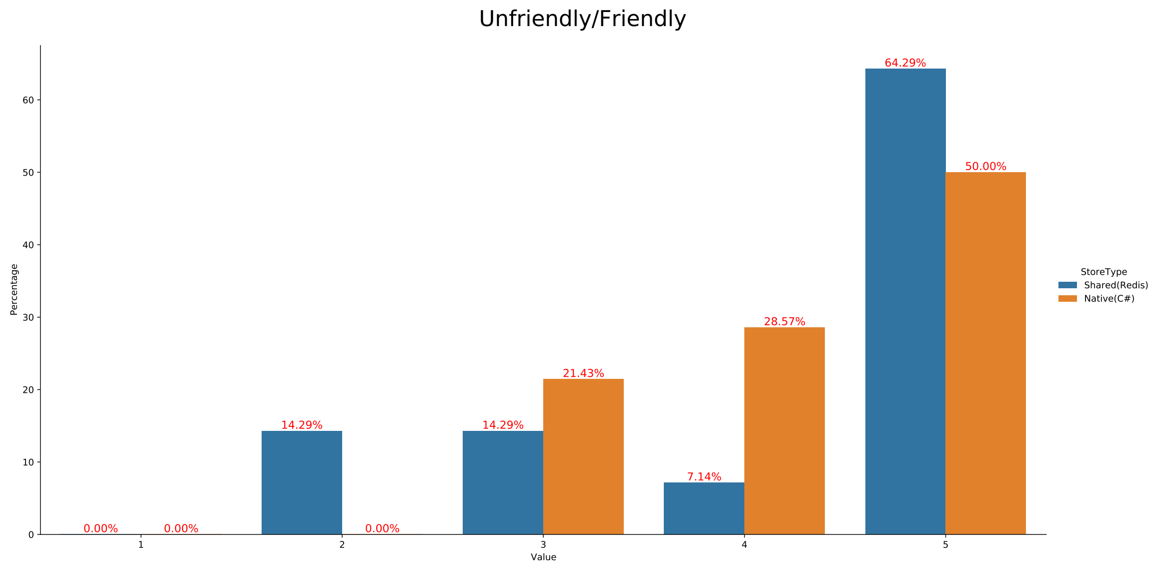

Figure A.15: X-axis: Participant ratings from 1: Unfriendly to 5: Friendly. Y-axis: the % of participants that selected those responses

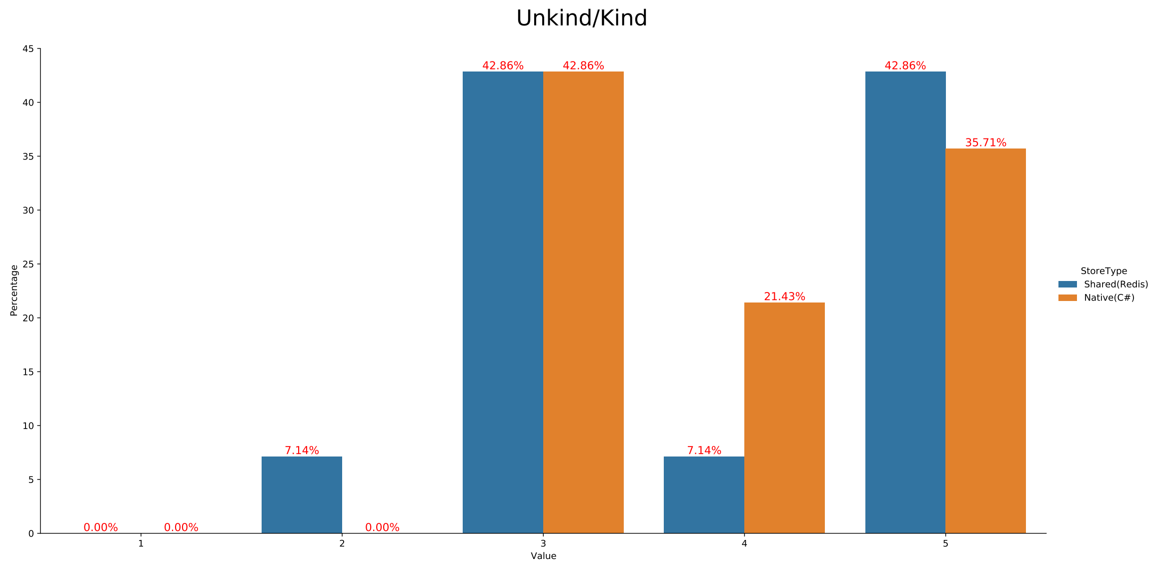

Figure A.16: X-axis: Participant ratings from 1: Unkind to 5: Kind. Y-axis: the % of participants that selected those responses

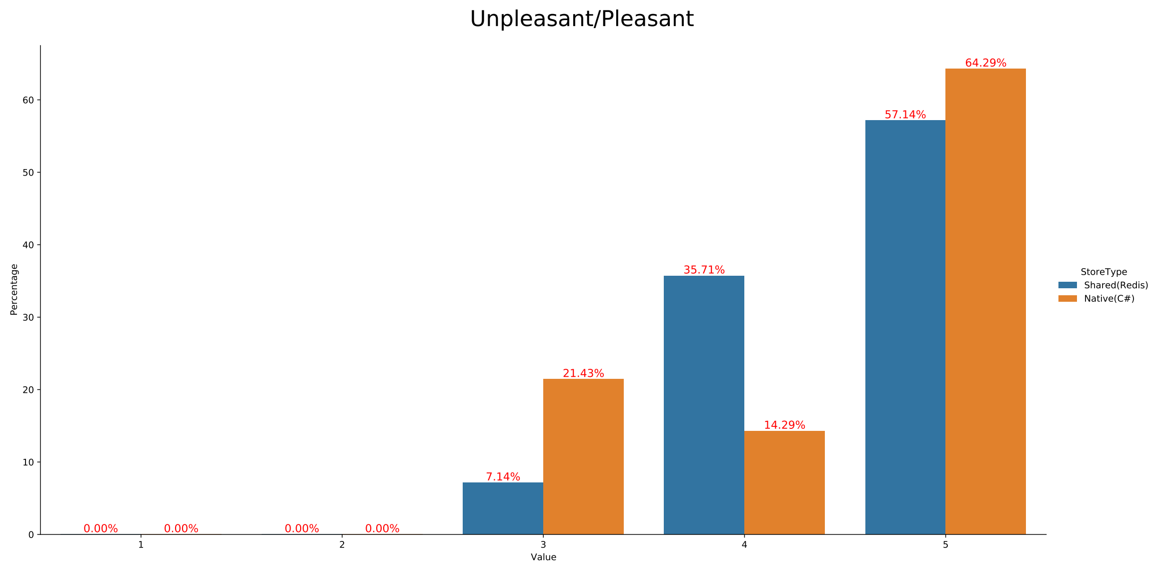

Figure A.17: X-axis: Participant ratings from 1: Unpleasant to 5: Pleasant. Y-axis: the % of participants that selected those responses

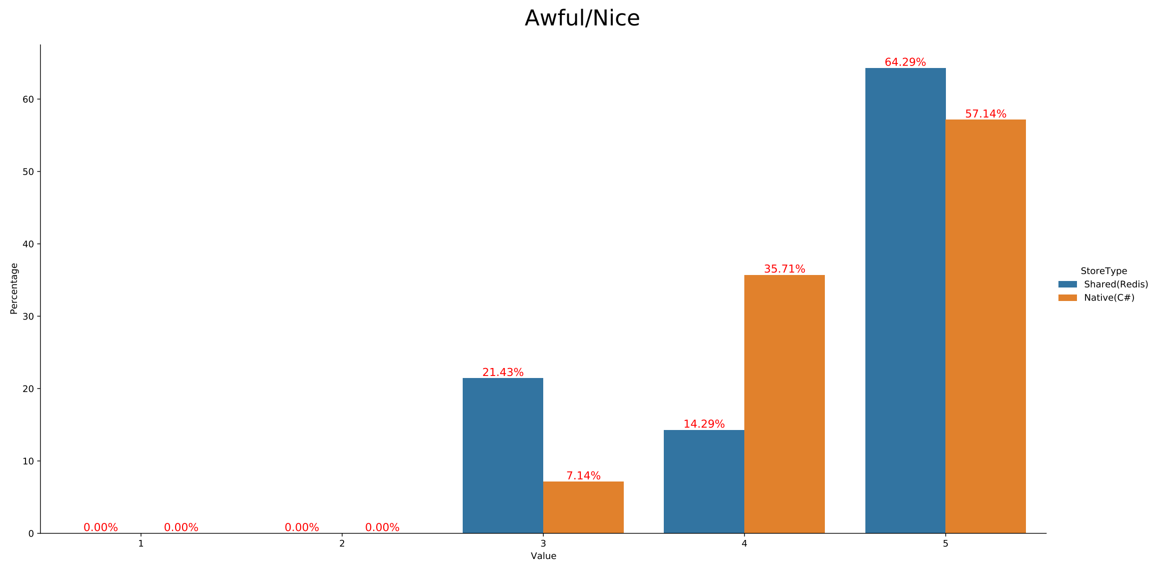

Figure A.18: X-axis: Participant ratings from 1: Awful to 5: Nice. Y-axis: the % of participants that selected those responses

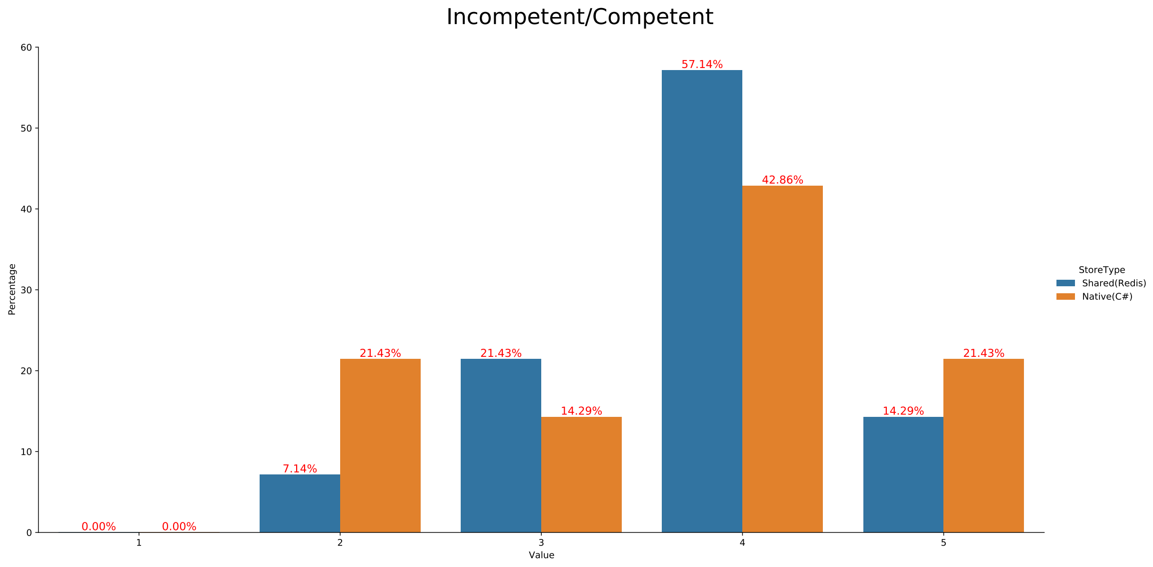

Figure A.19: X-axis: Participant ratings from 1: Incompetent to 5: Competent. Y-axis: the % of participants that selected those responses

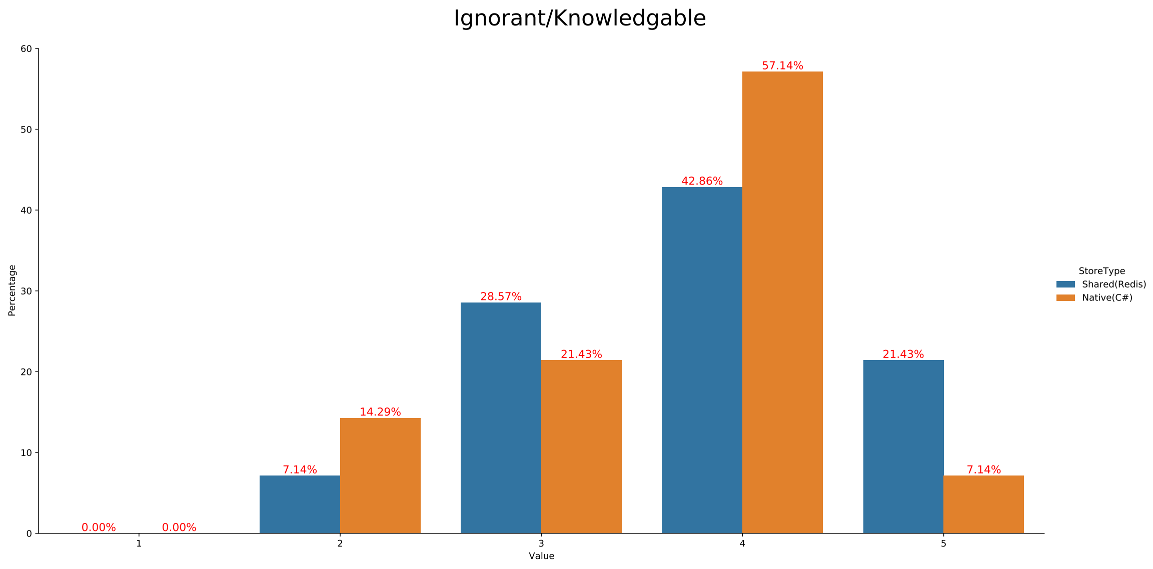

Figure A.20: X-axis: Participant ratings from 1: Ignorant to 5: Knowledgeable. Y-axis: the % of participants that selected those responses

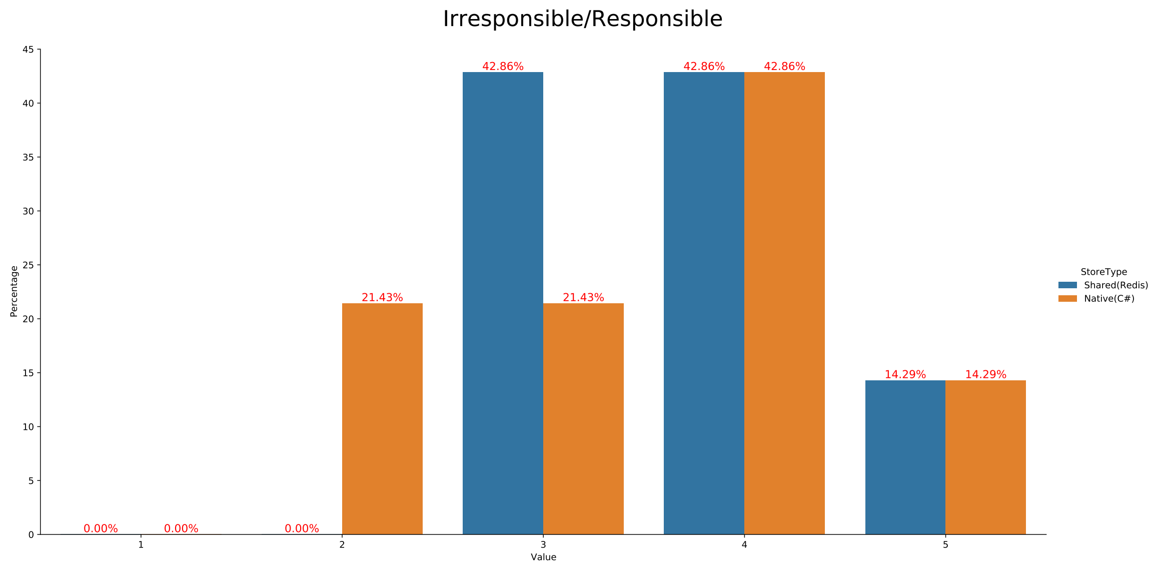

Figure A.21: X-axis: Participant ratings from 1: Irresponsible to 5: Responsible. Y-axis: the % of participants that selected those responses

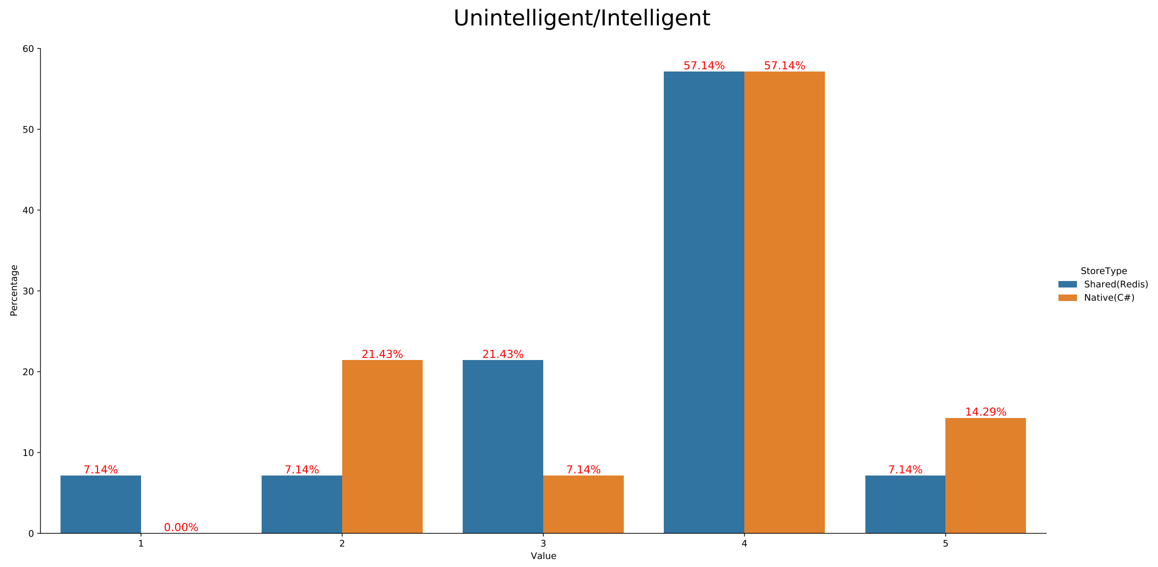

Figure A.22: X-axis: Participant ratings from 1: Unintelligent to 5: Intelligent. Y-axis: the % of participants that selected those responses

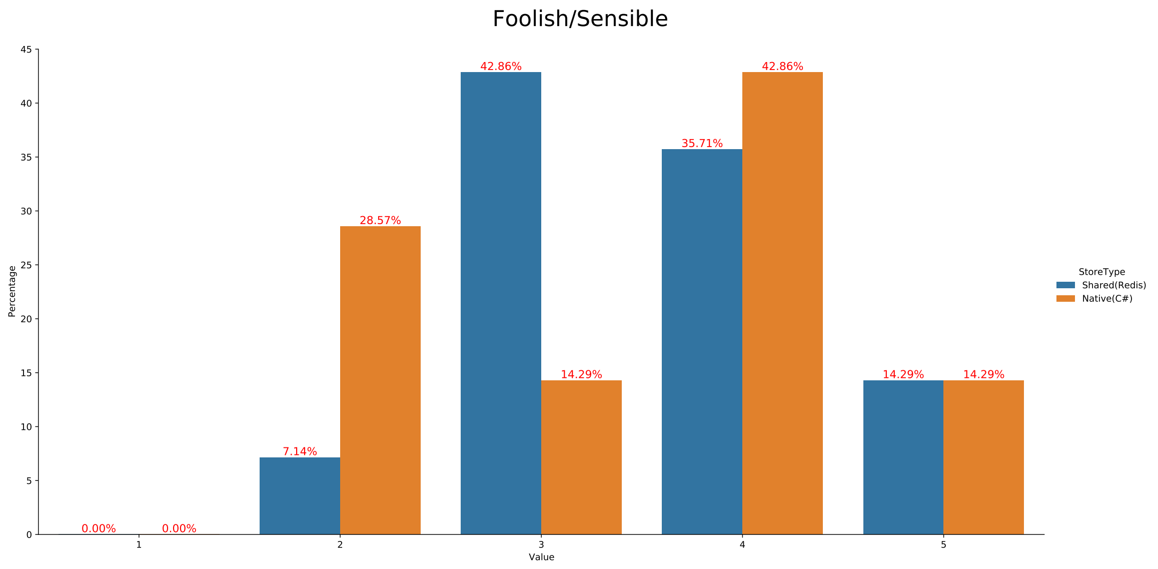

Figure A.23: X-axis: Participant ratings from 1: Foolish to 5: Sensible. Y-axis: the % of participants that selected those responses

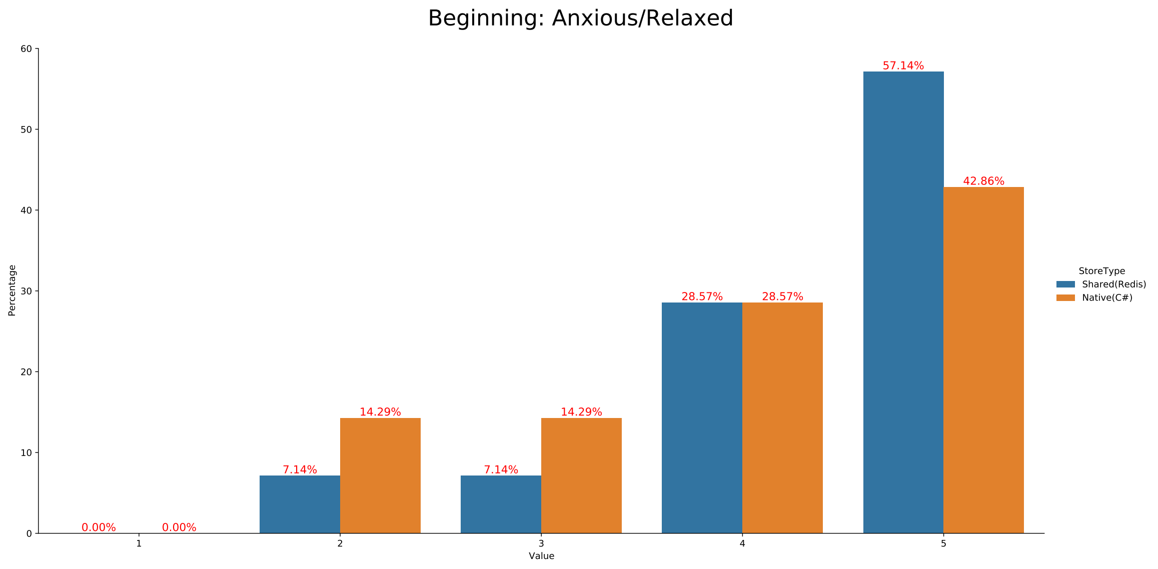

Figure A.24: X-axis: Participant ratings from 1: Anxious to 5: Relaxed. Y-axis: the % of participants that selected those responses

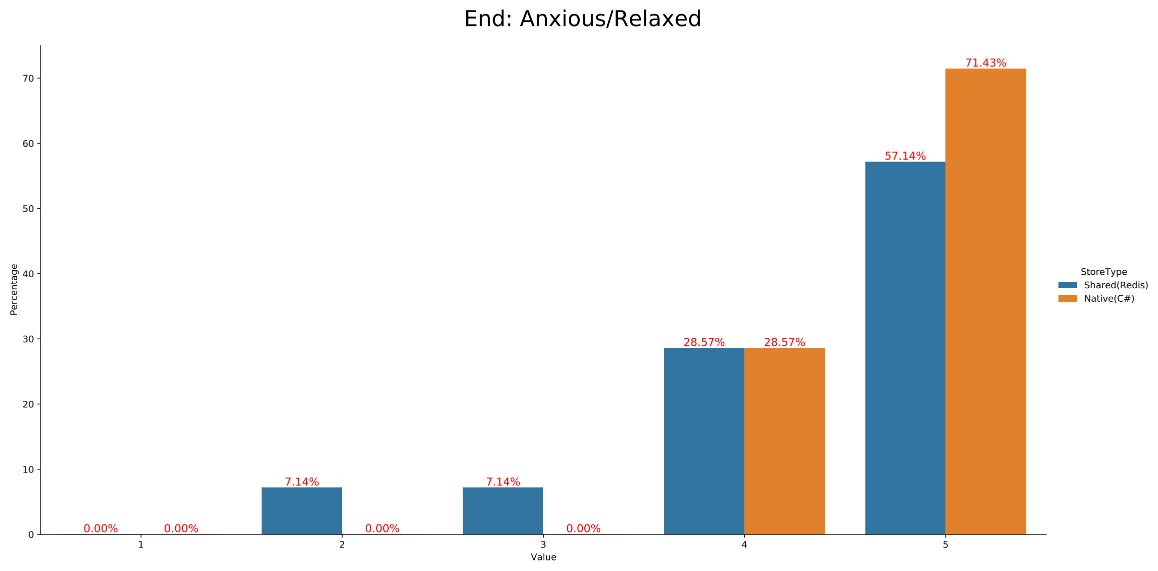

Figure A.25: X-axis: Participant ratings from 1: Anxious to 5: Relaxed. Y-axis: the % of participants that selected those responses

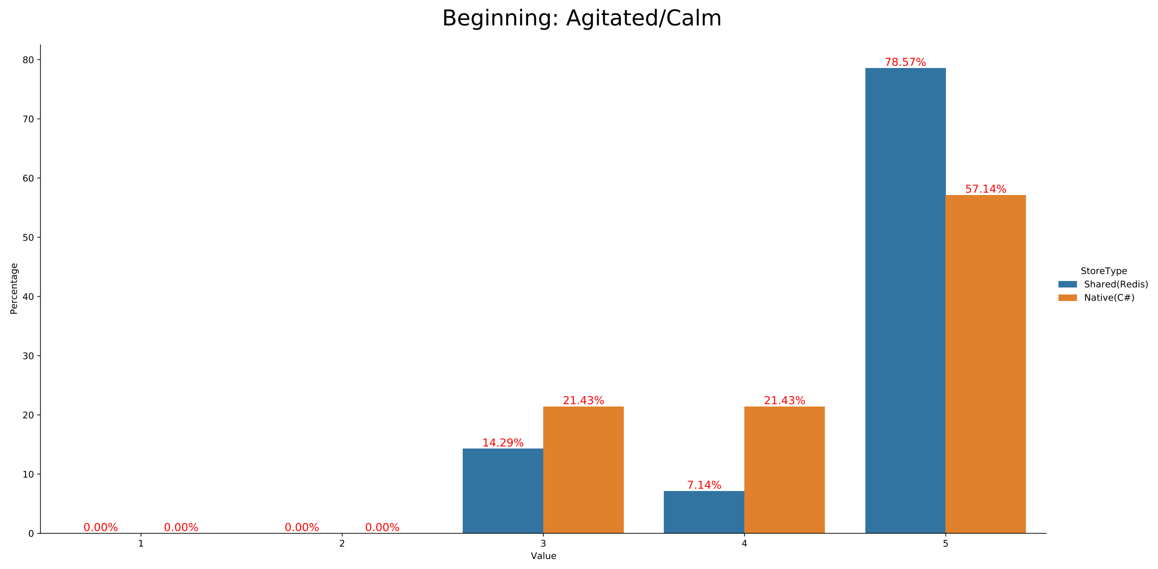

Figure A.26: X-axis: Participant ratings from 1: Agitated to 5: Calm. Y-axis: the % of participants that selected those responses

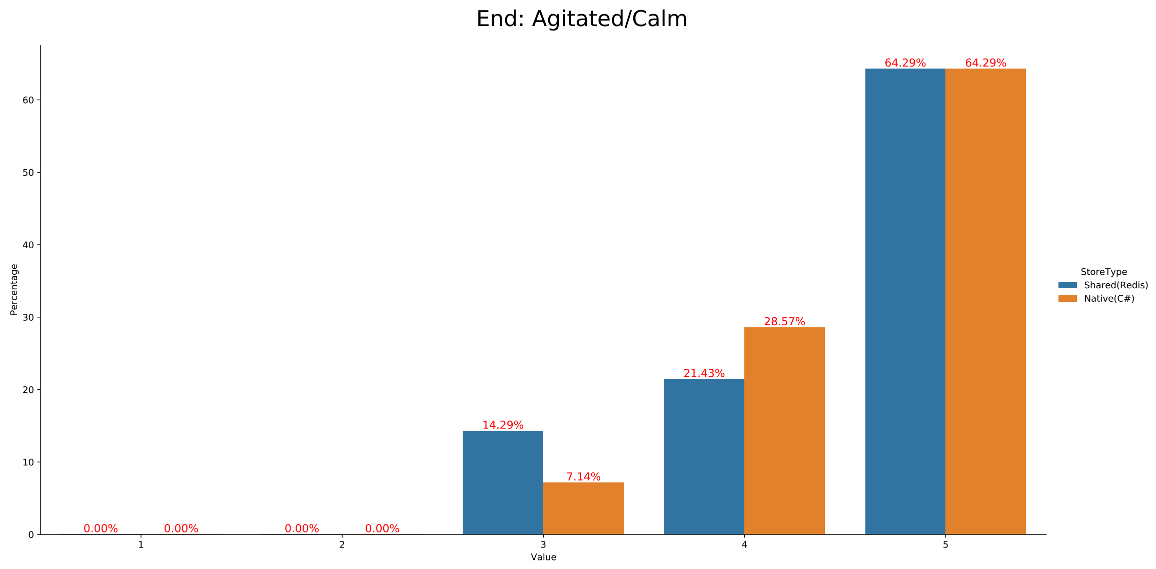

Figure A.27: X-axis: Participant ratings from 1: Agitated to 5: Calm. Y-axis: the % of participants that selected those responses

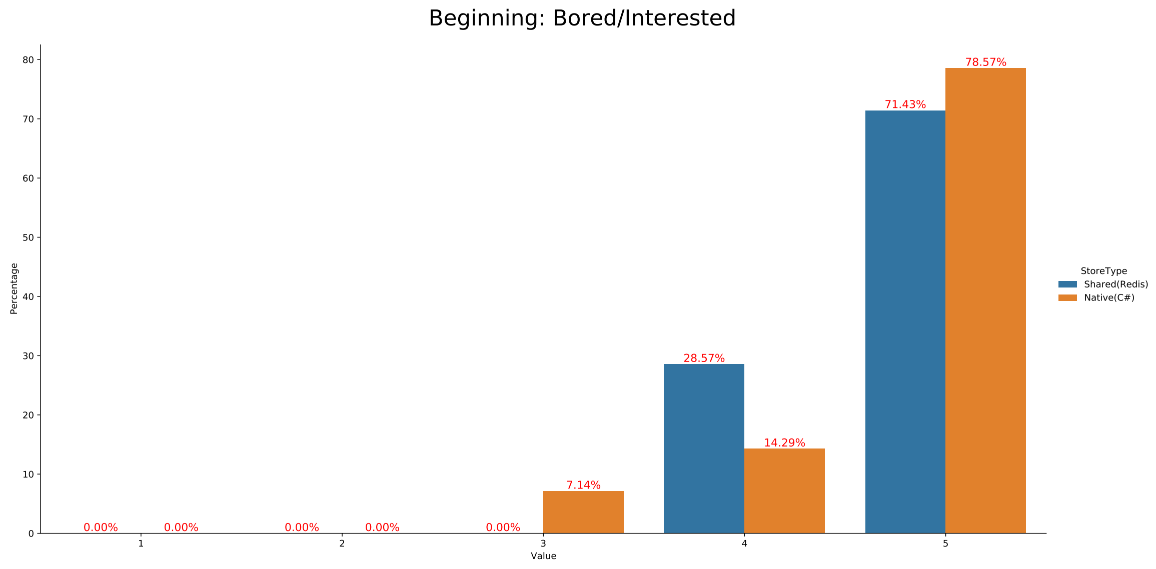

Figure A.28: X-axis: Participant ratings from 1: Bored to 5: Interested. Y-axis: the % of participants that selected those responses

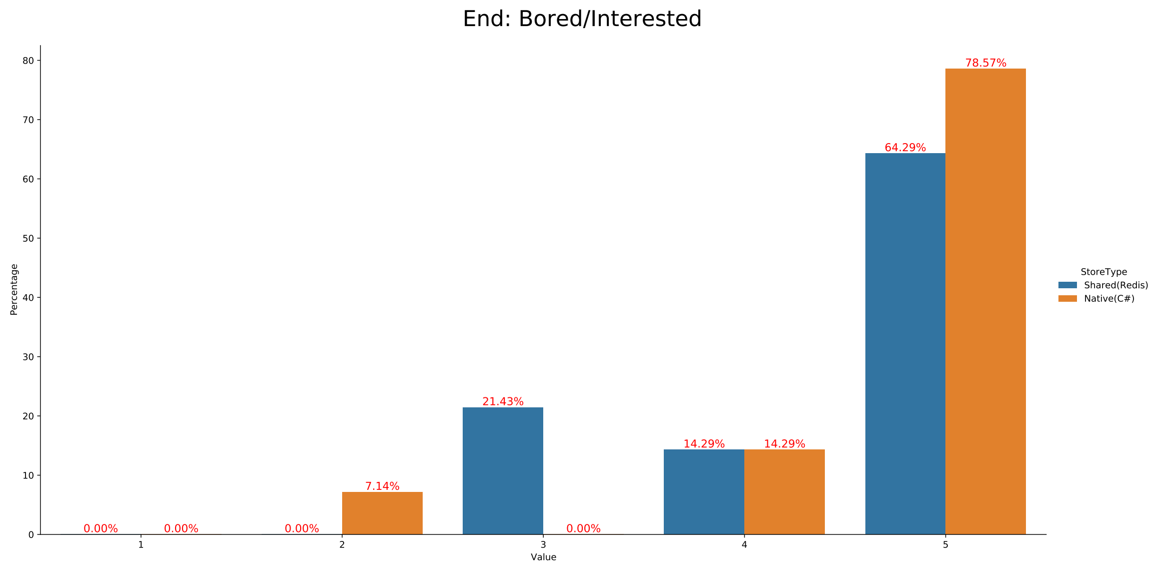

Figure A.29: X-axis: Participant ratings from 1: Bored to 5: Interested. Y-axis: the % of participants that selected those responses

APPENDIX B:

AUGMENTED GODSPEED QUESTIONNAIRE

Questionnaire

1. How attached to the robot did you feel? Mark only one oval.

Not at all 1 2 3 4 5 Very

2. How interesting was the robot to interact with? Mark only one oval.

Not at all 1 2 3 4 5 Very

3. Would you like to spend more time with the robot? Mark only one oval.

Not at all 1 2 3 4 5 Very much

4. Read the statement below and select one of the given options: The robot had a goal. Mark only one oval.

Yes No

5. If you agreed, what do you think the robot's goal was? Why do you think that?

6. If you disagreed, why do you disagree? What do you think robot was doing?

7. How many years old do you think the robot is (in terms of its behavior)?

8. Please rate your impression of the robot on this scale: Mark only one oval.

Fake 1 2 3 4 5 Natural

9. Please rate your impression of the robot on this scale: Mark only one oval.

Machinelike 1 2 3 4 5 Humanlike

10. Please rate your impression of the robot on this scale: Mark only one oval.

Unconscious 1 2 3 4 5 Conscious

11. Please rate your impression of the robot on this scale: Mark only one oval.

Artificial 1 2 3 4 5 Lifelike

12. Please rate your impression of the robot on this scale: Mark only one oval.

Moving rigidly 1 2 3 4 5 Moving elegantly

13. Please rate your impression of the robot on this scale: Mark only one oval.

Dead 1 2 3 4 5 Alive

14. Please rate your impression of the robot on this scale: Mark only one oval.

Stagnant 1 2 3 4 5 Lively

15. Please rate your impression of the robot on this scale: Mark only one oval.

Mechanical 1 2 3 4 5 Organic

16. Please rate your impression of the robot on this scale: Mark only one oval.

Inert 1 2 3 4 5 Interactive

17. Please rate your impression of the robot on this scale: Mark only one oval.

Apathetic 1 2 3 4 5 Responsive

18. Please rate your impression of the robot on this scale: Mark only one oval.

Dislike 1 2 3 4 5 Like

19. Please rate your impression of the robot on this scale: Mark only one oval.

Unfriendly 1 2 3 4 5 Friendly

20. Please rate your impression of the robot on this scale: Mark only one oval.

Unkind 1 2 3 4 5 Kind

21. Please rate your impression of the robot on this scale: Mark only one oval.

Unpleasant 1 2 3 4 5 Pleasant

22. Please rate your impression of the robot on this scale: Mark only one oval.

Awful 1 2 3 4 5 Nice

23. Please rate your impression of the robot on this scale: Mark only one oval.

Incompetent 1 2 3 4 5 Competent

24. Please rate your impression of the robot on this scale: Mark only one oval.

Ignorant 1 2 3 4 5 Knowledgable

25. Please rate your impression of the robot on this scale: Mark only one oval.

Irresponsible 1 2 3 4 5 Responsible

26. Please rate your impression of the robot on this scale: Mark only one oval.

Unintelligent 1 2 3 4 5 Intelligent

27. Please rate your impression of the robot on this scale: Mark only one oval.

Foolish 1 2 3 4 5 Sensible

28. At the BEGINNING of the interaction, how did you feel on this scale: Mark only one oval.

Anxious 1 2 3 4 5 Relaxed

29. At the END of the interaction, how did you feel on this scale: Mark only one oval.

Anxious 1 2 3 4 5 Relaxed

30. At the BEGINNING of the interaction, how did you feel on this scale: Mark only one oval.

Agitated 1 2 3 4 5 Calm

31. At the END of the interaction, how did you feel on this scale: Mark only one oval.

Agitated 1 2 3 4 5 Calm

32. At the BEGINNING of the interaction, how did you feel on this scale: Mark only one oval.

Bored 1 2 3 4 5 Interested

33. At the END of the interaction, how did you feel on this scale: Mark only one oval.

Bored 1 2 3 4 5 Interested

34. Of the following relations, which do you feel describe the robot best? Mark only

one oval.

Brother or Sister

Classmate

Stranger

Relative (e.g., cousin or aunt)

Friend

Parent

Teacher

Neighbor